Reconstrucción de una video-aventura isométrica clásica utilizando *IsoUnity* como herramienta de producción multiplataforma

Antonio Santamaría Barcina
Alejandro Alexiades Estarriol

GRADO EN INGENIERÍA DEL SOFTWARE
FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

TRABAJO DE FIN DE GRADO
DE INGENIERÍA DEL SOFTWARE

Madrid, 14 de septiembre de 2015

Director: Federico Peinado Gil
Co-director: Ismael Sagredo Olivenza

Autorización de difusión y utilización

Antonio Santamaría Barcina y Alejandro Alexiades Estarriol autorizamos a la Universidad Complutense de Madrid a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la propia memoria, como el código, los contenidos audiovisuales incluso si incluyen imágenes de los autores, la documentación y/o el prototipo desarrollado.

Fdo. Antonio Santamaría Barcina

Fdo. Alejandro Alexiades Estarriol

Agradecimientos

Antonio:

Quiero dar gracias a los “gemelos” (Desarrolladores de IsoUnity), por toda la ayuda que me han dado para entender su herramienta. Siempre han estado ahí cuando me ha surgido alguna duda o para resolver algún bug localizado, a pesar de que mi horario de programación no es muy… normal.

A Ismael por enseñarme lo que es Unity y explicarme ciertos detalles de programación que demuestran porque yo hago cosas que “funcionan” mientras que él programa.

A Federico, quien puede darte más ideas por minuto de las que puedes desarrollar en dos años de trabajo. Su conocimiento sobre videojuegos e ideas claras y adaptables han hecho posible este proyecto.

Y por último a Alejandro por el entusiasmo mostrado por parte de un Gamer de pura cepa.

Ha sido un placer.

Alejandro:

Me gustaría hacer mención de lo fácil que ha sido trabajar y coordinarse con mi compañero Antonio. Desde el principio ha habido una motivación por hacer un videojuego con la mayor calidad posible. Que es una de las razones por la cual hayamos decidido extender el proyecto dos meses más de desarrollo.

Índice de contenidos

Agradecimientos 2
Índice de figuras 5
Resumen 7
Abstract 9
Capítulo 1.- Introducción 11
1.1.- La industria de los videojuegos 12
1.2.- Los videojuegos clásicos y su reconstrucción 14
1.3.- Estructura del trabajo 15
Capítulo 2.- La video-aventura isométrico clásico y la producción de videojuegos 17
2.1.- Video-aventuras isométricas 17
2.1.1.- Gráficos isométricos 17
2.1.2.- Historia de los juegos isométricos 19
2.1.3.- Ejemplos 21
2.2.- La Abadía del Crimen 26
2.2.1.- Creadores de La Abadía del Crimen 26
2.2.2.- Versiones de La Abadía del Crimen 27
2.3.- Herramientas de desarrollo 30
2.3.1.- Unreal Engine 31
2.3.2.- CryEngine 31
2.3.3.- RPG Maker 31
2.3.4.- Filmation Game Creator 31
2.3.5.- Nimrod 32
2.3.6.- Isogenic Game Engine 33
2.3.7.- Flexible Isometric Free Engine 33
2.3.8.- Unity 34
2.3.9.- *IsoUnity* 36
2.4.- Elaboración de un videojuego 37
2.4.1.- Roles 39
2.4.2.- Ciclo de Vida 39
2.4.3.- Guías 42
Capítulo 3.- Objetivos y metodología 43
3.1.- Motivaciones del proyecto 43
3.2.- Objetivos 43

3.3.- Plan de trabajo 44
3.4.- Progresos previstos para *IsoUnity* 51
Capítulo 4.- *IsoAbbey*: Análisis y diseño de una video-aventura clásica 52
4.1.- Análisis de La Abadía del Crimen 52
4.1.1.- Desarrollo de la trama del juego 53
4.1.2.- Acciones generales de los personajes 53
4.1.3.- Jugabilidad 57
4.2.4. Cámara del juego, diseño y ambientación. 63
4.1.5.- Personajes de *La Abadía del Crimen* 66
4.1.6.- Extracción de texturas de *La Abadía del Crimen* 67
Capítulo 5.- *IsoAbbey*: Implementación multiplataforma con *IsoUnity* 68
5.1.- Uso práctico de *IsoUnity* y prueba 68
5.2.- Construcción de escenarios en *IsoUnity* 68
5.3.- Características nuevas 73
5.4.- Elaboración del control del juego 73
5.4.1.- Día 1 en *La Abadía del Crimen* 74
5.4.2.- Elementos implementados necesarios para *IsoAbbey* 80
Capítulo 6.- Discusión sobre *IsoAbbey* y el futuro de *IsoUnity* 86
6.1.- Problemas de usabilidad de *IsoUnity* 86
6.2.- Bugs encontrados en el desarrollo 87
6.3.- Problemas y requisitos adicionales encontrados 90
6.4.- *IsoUnity* en un entorno de producción 91
Capítulo 7. Conclusiones 93
7.1.- Conclusiones 93
7.2.- Aportaciones de los integrantes 94
7.1.1.- Antonio Santamaría Barcina 94
7.1.2.- Alejandro Alexiades Estarriol 95
Bibliografía 97
Anexos 100
Anexo 1.- Title (in English) 100
Anexo 2.- Conclusions (in English) 101
Anexo 3.- Carta de Risin´Goat 103

Índice de figuras

Figura 1 Gráficos del crecimiento en la facturación de un 130% previsto para los próximos años 12
Figura 2 Gráfico de la evolución del empleo prevista hasta 2018 (DEV, 2015) 13
Figura 3 Gráfico de la distribución de la facturación por tipo de modelo 13
Figura 4 Comparativa entre la portada del juego E.T. The Extra-Terrestrial y una captura de pantalla real en Atari 2600 14
Figura 5Visión Isométrica respecto al plano y ángulos que hay que generar. 18
Figura 6 Ant Attack 18
Figura 7 Juegos isométricos en los 90 y adaptaciones 19
Figura 8 Videojuegos isométricos de estrategia y RPG 20
Figura 9 Contraste de imágenes entre el juego original del Knight Lore y su Remake 22
Figura 10 Sistema de ecuaciones de Christopher Jon definidas en el documento Knight Lore data format 22
Figura 11 Diferentes vistas de una misma habitación del Knight Lore 2006 y el original 23
Figura 12 Contraste de imágenes entre el juego original Batman y su Remake 24
Figura 13 Contraste de imágenes entre el juego original Head Over Heels y su Remake 24
Figura 14 Contraste de imágenes entre el juego original Alien 8 y su Remake 25
Figura 15 Remake de La Abadía del Crimen de Antonio Giner 27
Figura 16 Remake de la Abadía del Crimen de Manuel Pazos y Armando Pérez 28
Figura 17 Remake 3D de La Abadía del Crimen llamada The Abbey 28
Figura 18 Remake de La Abadía del Crimen de Manuel Pazos y Daniel Celemín 30
Figura 20 Captura de la herramienta de Filmation Game Creator 32
Figura 21 Captura de la herramienta Nimrod 32
Figura 22 Captura de la herramienta Isogenic Game Engine 33
Figura 23 Captura de la herramienta Flexible Isometric Free Engine 34
Figura 19 Captura de la herramienta IsoUnity 37
Figura 24 Fases del proceso de desarrollo. 41
Figura 25 Escala de tiempo de las tres primeras fases RUP del proyecto 44
Figura 26 Diagramas de Gantt para programar las tareas del proyecto 45
Figura 27 Diagrama de clases del editor de mapas 47
Figura 28 Diagrama de secuencia del funcionamiento general de los distintos editores del mapa 48
Figura 29 Gráfico de esfuerzo a lo largo de la fase de elaboración 50
Figura 30 Metodología empleada en el proyecto 50
Figura 31 Herramienta Waffle para el control de objetivos 51
Figura 32 Panel de control del juego en el pie de página del juego 58
Figura 33 Mensajes sobre una estética de pergamino en el juego 60
Figura 34 Plano original de La Abadía del Crimen en el que se basa el mapa desarrollado para nuestro Remake 63
Figura 35 Introducción inicial al lanzar la partida (Prólogo) 64
Figura 36 Cámara ortogonal en la imagen de la izquierda y perspectiva 3D libre en la de la derecha 64
Figura 37 Estancia de nuestro Remake y misma imagen en el juego original a la derecha 65

Figura 38 Diseño original de Guillermo y Adso 66
Figura 39 Plantilla de personajes que utiliza IsoUnity para crear los sprites 66
Figura 40 Herramienta Texture Assistant 67
Figura 41 Vista de una cell en el entorno de desarrollo Unity 68
Figura 42 Cells con distintas alturas y acabados superiores 69
Figura 43 Aplicación de diversas texturas en las Cells anteriores 70
Figura 44 Comparación de un mapa creado en IsoUnity (sin utilizar decoraciones) con un mapa de un videojuego real 70
Figura 45 Habitación realizada en IsoUnity que incluye diversas decoraciones 71
Figura 46 Varios personajes (Entities) en un mismo mapa 72
Figura 47 Esquema de arquitectura empleada 73
Figura 48 Prólogo del videojuego 74
Figura 49 Estados pertenecientes a la máquina principal 76
Figura 50 Estados pertenecientes a la máquina de los monjes 80
Figura 51 Función Movimiento libre 81
Figura 52 Función perseguir 81
Figura 53 Función patrullar 82
Figura 54 Comparación de Hud entre nuestro juego y el original 82
Figura 55 Imagen de los carteles mostrados en el juego 83

Resumen

Actualmente el videojuego es uno de los medios de entretenimiento que mayor crecimiento está experimentando tanto en número de usuarios como en cifras de negocio en los últimos años. Entre los jugadores existe una fuerte tendencia a recordar y volver a jugar títulos clásicos de la historia de los videojuegos, o títulos "retro" que recuerdan aspectos de dichos juegos de épocas pretéritas. Cada vez es más habitual que se realicen reconstrucciones ("remakes") de juegos antiguos para que puedan jugarse en plataformas actuales e incorporen a menudo características con las que no contaba el original, pero que tienen una gran demanda en la actualidad. Uno de los géneros más queridos por los gamers es el de la video-aventura, que en los años 80 usaba a menudo escenarios estructurados en casillas o bloques vistos en perspectiva isométrica. En España se desarrolló una obra emblemática de este tipo de juegos, *La Abadía del Crimen*, un título que todavía es muy popular en nuestros días.

Conocido el interés de algunos estudios de producción por reconstruir de nuevo esta video-aventura y adaptarla a las necesidades de nuestros tiempos, para este trabajo nos propusimos realizar un estudio sobre la viabilidad de desarrollar semejante proyecto a nivel profesional. Buscábamos plantear un diseño y una funcionalidad mejorados, así como contar con la capacidad de ser jugado de manera cómoda e intuitiva en un conjunto muy amplio de plataformas (sobremesa, navegador web, tabletas y móviles inteligentes).

Nuestro trabajo comienza con una revisión de la historia del videojuego, con especial énfasis en las video-aventuras clásicas. Tras esto, se analizan las metodologías más habituales en la industria para el diseño y producción de videojuegos, que a menudo son metodologías ágiles de gestión de proyectos. También se exploran las herramientas más comunes para el desarrollo de este tipo de software, haciendo hincapié en aquellas más específicas, la cuales permiten crear escenarios estructurados en casillas o bloques vistos desde una perspectiva isométrica y cuyos gráficos tienen un aspecto "pixelado" tan característico de la baja resolución de imagen que ofrecían las pantallas de los años 80.

Tras definir con mayor precisión los objetivos de este estudio y el plan de trabajo a seguir, se explican los fundamentos en lo que se basa la propuesta de trabajo. Ésta se fundamenta en utilizar *IsoUnity*, una herramienta experimental de origen universitario que trata de facilitar la creación de este tipo de juegos "retro" de vista isométrica y jugabilidad de video-aventura clásica, y que está integrada en Unity, un entorno profesional de desarrollo de videojuegos ampliamente utilizado en nuestros días.

Como parte central de este trabajo, se muestra la especificación y las ideas detrás del nuevo diseño aplicado, tratando de aclarar cómo se afrontan las peculiaridades del juego original que deben ser adaptadas a las nuevas tecnologías y tendencias en materia de videojuegos. Se describe el prototipo que se ha implementado, explicando cómo ha sido la experiencia de uso de *IsoUnity* y el proceso de extender parte de su código para

habilitar comportamientos más complejos en los personajes. Se detalla cómo funciona la herramienta ante las pruebas a las que se expone que pueden simular una carga equivalente a la de un videojuego real.

Finalmente, se presenta una discusión acerca de la problemática que plantea la reconstrucción de videojuegos clásicos, las ventajas e inconvenientes encontrados durante el proceso, tanto con el rediseño de *La Abadía del Crimen* como con el uso de *Unity* e *IsoUnity* como plataformas de desarrollo. Como conclusión a este trabajo, consideramos viable que un estudio de producción aborde de esta manera el proyecto y trate de convertirlo en un producto comercial, siempre que se tengan presentes una serie de recomendaciones relativas a su alcance que enumeramos al final.

Palabras clave: Videojuegos, Motores de videojuegos, Unity, Extensiones de motores de videojuegos, Videojuegos en perspectiva isométrica, *La Abadía del Crimen*, *IsoAbbey*

Abstract

Currently the videogame is one of the entertainment ways which experiencing the greatest growth in number of users and business figures. Among gamers there is a strong tendency to remember and return to play classic titles in the history of videogames, or “retro" titles that recall aspects of these games of earlier times. It is increasingly common the reconstruction ("remakes") of old games that are made so they can be played on existing platforms and often incorporate features that the original did not have, but are in high demand today. One of the most beloved genres for gamers is the video adventures, which in the 80's used often structures in boxes or blocks scenarios seen in isometric perspective. In Spain an emblematic work of these games, The Abbey of Crime, a title that is still very popular today was developed.

Knowing the interest of some production studios to rebuild again this video adventures and adapt it to the needs of our times, we set out for this work a study on the feasibility of developing such a project professionally. We wanted to raise design and improved functionality as well as have the ability to be played comfortably and intuitively in a wide range of platforms (desktop, web browser, tablets and smart phones).

Our work begins with a review of gaming history, with special emphasis on the classical video adventures. Afterwards we analyze the most common methodologies in the industry for the design and production of video games, which often are agile methodologies of project management. We also explored the most common tools to develop this type of software, emphasizing more specific ones, which allow to create scenarios structured in boxes or blocks seen in isometric perspective and whose graphics have a “pixelated" look so typical of the low resolution image featuring screens in 80’s.

After defining with accuracy the objectives of this study and the work plan to follow, we explained that our proposal is based on use of *IsoUnity*, an experimental university-based tool that aims to facilitate the creation of such "retro" games of isometric type and video adventure classic gameplay, and that is integrated in Unity, a professional game development environment widely used today.

As a central part of this work we show the specification and the ideas behind our new design, trying to clarify how the peculiarities of the original game are to be adapted to new technologies and trends in video games. We describe the prototype we have implemented, explaining how it has been the experience of using *IsoUnity* and the process of extending some of its code to enable more complex behaviors in the characters. We detail load tests conducted to validate the scalability of the tools used and also the possibility of implementing with them all the features contained in our specification of the new video game.

Finally it is presented a discussion about the problems posed by the reconstruction of classic video games, the advantages and disadvantages encountered during the process, both with the redesign of The Abbey of Crime as to the use of Unity and *IsoUnity* as development platforms. As a conclusion to this paper, we consider feasible that a production studio approaches in this way the project and try to turn it into a commercial product, provided that a series of recommendations for its scope, which are listed at the end are taken into account.

Keywords: Games, Game Engines, Unity, Game Engine Plugins, Isometric games, *La Abadía del Crimen*, *IsoAbbey*

Capítulo 1.- Introducción

El ocio, ese interés tan propio del ser humano, que se basa en actividades que le permitan divertirse y olvidarse temporalmente de la rutina del trabajo, vivió una gran expansión durante la Revolución Industrial. En esa época floreció la que hoy llamamos Industria del Entretenimiento, en parte como respuesta a la alienación que se vivía en las cadenas de producción. Aunque solemos asociar el entretenimiento con las artes tradicionales, como la Música, la Literatura o el Teatro, con el desarrollo tecnológico surgieron nuevas formas de divertirse como el Cine, la Televisión o los Videojuegos.

Los videojuegos son aplicaciones software multimedia capaces de aprovechar la interactividad con sus usuarios para suscitar emociones en ellos, hacerles sentir inmersión, asombro, excitación, y espolear su curiosidad para aprender y superarse a sí mismos. Precisamente sobre los videojuegos y la emoción que nos generan, tanto en las plataformas de hoy día como en los ordenadores personales de los que disponíamos hace décadas, es sobre lo que trata este trabajo.

Nuestro punto de partida es una conversación con uno de los jóvenes estudios de producción de videojuegos que están apareciendo en España en los últimos años. El estudio se estaba planteando recrear uno de los juegos clásicos de los años 80, llamado *La Abadía del Crimen*, y convertirlo en un producto capaz de funcionar en el mercado actual. Las dudas surgían tanto a nivel de diseño como a nivel técnico y de implementación, y les propusimos realizar nosotros esa labor de investigación previa para averiguar si un proyecto de estas características sería viable.

Este trabajo debe comenzar por la revisión de la historia de los videojuegos prestando atención a las video-aventuras clásicas del tipo particular que nos interesa y a sus reconstrucciones, en el caso de que hayan existido. A continuación, debemos conocer las distintas herramientas de desarrollo de videojuegos que existen, en particular las que sean más apropiadas para el tipo de juego a realizar (como veremos más adelante, aquí vamos a tratar con *IsoUnity*, una extensión de la plataforma *Unity*). De igual manera, se debe establecer una metodología de trabajo y adaptarla a las necesidades concretas de este proyecto. Como en cualquier otro proyecto de Ingeniería del Software, tendremos que analizar el problema al que nos enfrentamos y extraer unos requisitos funcionales y no funcionales (un análisis sobre el mismo), siendo en este caso las características del videojuego a reconstruir. Seguiremos un proceso de diseño, implementación y pruebas que culminará con una reflexión al final del trabajo sobre el resultado obtenido y las ventajas e inconvenientes encontrada en este proceso. Las conclusiones de este trabajo deberán aportar información útil a todos aquellos que pretendan acometer un proyecto de reconstrucción y actualización de un videojuego clásico para el público actual.

En las próximas secciones de este capítulo se introducirán algunas ideas sobre el videojuego y su industria, donde se empezará a hablar sobre los videojuegos clásicos que consolidaron esta forma de ocio en la década de los 80 y sobre la tendencia actual, algo nostálgica, de los jugadores veteranos. Son ellos los que han demandado nuevas reconstrucciones de aquellos juegos y han avivado la moda "retro" sobre todo entre los estudios de producción independientes. Esto es relevante para explicar la motivación de

este trabajo, ya que toda nuestra propuesta gira en torno a la reconstrucción, con técnicas y herramientas modernas, de un clásico de los videojuegos.

Por último, se encuentra una sección donde se detalla la estructura que se ha seguido para organizar en capítulos el contenido de esta memoria.

1.1.- La industria de los videojuegos

Para comprender bien qué es un videojuego, tenemos que analizar los distintos elementos que lo configuran. Comenzaremos por su contenido, constituido principalmente por su apartado visual y sonoro. Después, hablaremos de su tratamiento en el sector industrial y finalmente de su repercusión en el mercado, a través del marketing de masas.

Uno de los aspectos que más ha evolucionado en la historia de los videojuegos es su apartado visual. En los inicios del videojuego, estos eran gráficamente más simples debido a que las plataformas hardware no disponían de la potencia gráfica que hay en la actualidad. Sin embargo, en estas dos últimas décadas, gracias a la evolución de los microprocesadores, los gráficos de los videojuegos son tridimensionales y muy ricos desde el punto de vista de la geometría, la iluminación, las animaciones y los efectos visuales, etc., y han dado lugar a infinidad de títulos que presentan escenarios de gran realismo. En cuanto a la banda sonora y los efectos sonoros, podemos decir que son fundamentales para recrear ambientes que tratan de conmover y emocionar al jugador.

Como mencionamos antes, el videojuego ha sido la última incorporación a la Industria del Entretenimiento. A pesar de su corta vida, se sitúa en la primera posición en cuanto a volumen de negocio (Newzoo, 2015). En España, la industria espera crecer en facturación a un ritmo del 23,7% anual en los próximos 3 años, llegando a alcanzar en 2018 los 998 millones de euros, según un estudio de la Asociación Española de Empresas Productoras y Desarrolladoras de Videojuegos y Software de Entretenimiento (DEV, 2015), ilustrado mediante el gráfico de la Figura 1. A estos resultados han contribuido el hecho de que cada vez se consuman más videojuegos para las nuevas plataformas, tales como los teléfonos móviles y las tabletas.

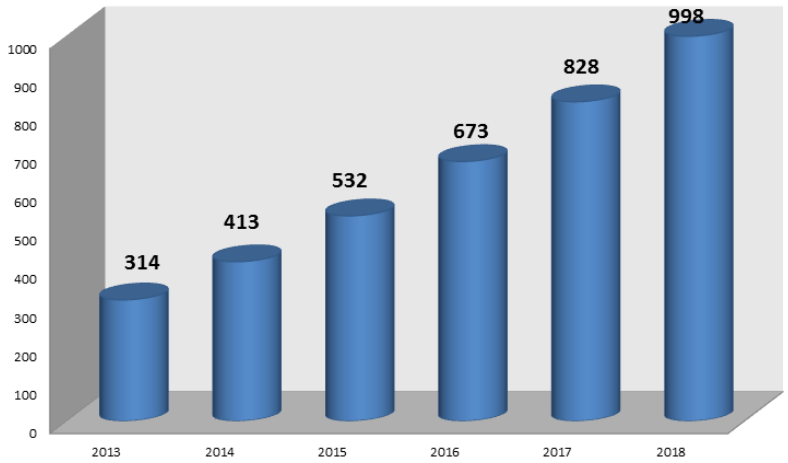


Figura 1 Gráficos del crecimiento en la facturación de un 130% previsto para los próximos años

Otro aspecto a tener en cuenta es el aumento de empleos en este sector. En 2013 aumentó un 28% el número de empleados y se contempla que crezca a un ritmo del 20,9% anual hasta 2018 (véase la Figura 2).

Figura 2 Gráfico de la evolución del empleo prevista hasta 2018 (DEV, 2015)

Finalmente, en la Figura 3 se puede apreciar cual es la tendencia actual en la venta de videojuegos. Las ventas en soporte digital ocupan casi el 90% del mercado relevando al soporte físico a un mísero 10%, al observar los modelos de venta de años anteriores se puede suponer que esta tendencia relegara al soporte físico a porcentajes meramente anecdóticos.

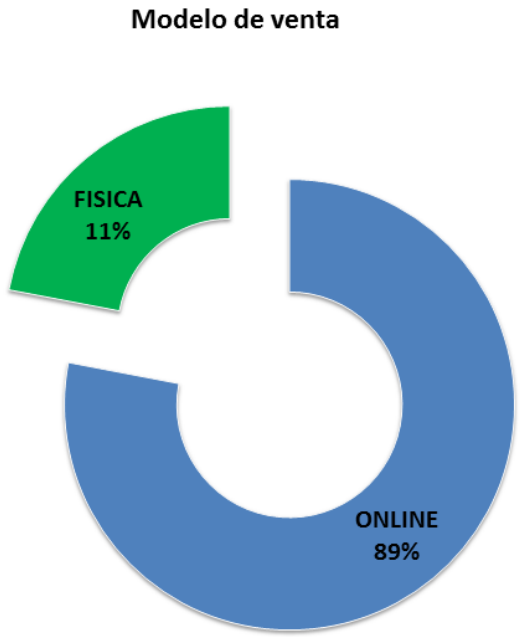


Figura 3 Gráfico de la distribución de la facturación por tipo de modelo

Hoy día casi cualquier dispositivo electrónico que se comercializa tiene la posibilidad de ejecutar aplicaciones y, por tanto, permite ejecutar videojuegos. El ejemplo más claro es el de los teléfonos móviles, que han conseguido que cada vez más personas de muy distintos perfiles disfruten de ellos.

Para finalizar este resumen, diremos que la distribución de un videojuego en el mercado supone una serie de procesos en los que el marketing juega un papel de gran importancia. La portada ha sido desde siempre un importante factor de decisión en el consumidor (Serrano Acosta, 2014). Esta debe transmitir la esencia del videojuego y, al mismo tiempo, hacerlo atractivo para el jugador y motivar su compra (véase la Figura 4). Hace 30 años, los gráficos del videojuego quedaban muy lejos de lo que podría crear un ilustrador profesional en la portada, aunque esto forma parte de la "magia" de aquella época.

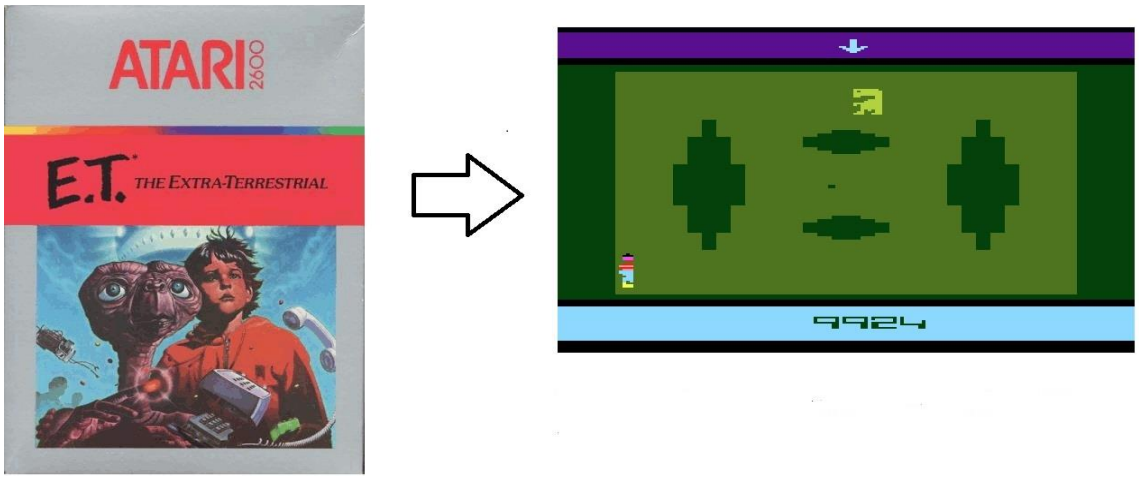


Figura 4 Comparativa entre la portada del juego E.T. The Extra-Terrestrial y una captura de pantalla real en Atari 2600

En los juegos de última generación, el marketing se centra más en mostrar videos ya sean vídeos promocionales (teasers) o tomados directamente del juego, que siempre resultan mucho más fieles a la experiencia real de jugarlo y disminuyen los miedos que están generando los primeros, que muestran algo que luego no puede verse "in Game".

1.2.- Los videojuegos clásicos y su reconstrucción

Desde sus inicios, los videojuegos han evolucionado con la intención de que el jugador se sienta cada vez más integrado e inmerso en el universo del juego. Por eso, en los años 80, aunque todavía no había tecnología disponible para representar juegos completamente en 3D, aparecieron los primeros videojuegos que usaban técnicas como la proyección isométrica que, mediante gráficos 2D, eran capaces de mantener cierta ilusión de tridimensionalidad.

Pero a medida que han transcurrido los años, desde la aparición de los primeros juegos 2D, ha surgido una sensación de nostalgia sobre estos primeros juegos que no se

esperaba. Muchas personas que pertenecen a esta generación de juegos clásicos, recuerdan aquellos días con mucho aprecio. Esta razón puede que sea la que influido más a la hora de desarrollar varias reconstrucciones sobre juegos clásicos. Incluso hay eventos como el Retro Madrid o Retro Barcelona, donde se reúnen desde profesionales en el sector de los videojuegos como aficionados, a compartir su pasión por los videojuegos clásicos.

En España se vivió la llamada "Edad de Oro del Software Español", entre 1983 y 1992 (Esteve, 2014), años en los que nuestro país llegó a ser, por detrás de Reino Unido, uno de los mayores productores europeos de software de entretenimiento para máquinas de 8 bits, en especial para el sistema doméstico Spectrum (Sinclair Research ,1982).

Fue a finales del 1983 cuando *Indescomp* realizó el lanzamiento de *La Pulga,* el primer videojuego español en lograr una distribución internacional considerable. Luego se hicieron una serie de adaptaciones para otras plataformas como *Commodore,* donde recibió el nombre de *Booga-Boo,* comercializándose también en Estados Unidos en máquinas como Amstrad con el nombre de *Roland in the Cave*.

Otro juego a destacar en esta década, es el título *La Abadía del Crimen*, del cual se entrará más en detalle en el capítulo 4. Este videojuego se caracteriza por tener un vistoso y original diseño en sus gráficos, y esto supuso que fuera considerado un videojuego adelantada a su época. Por este motivo el juego tuvo un gran número de seguidores, que posteriormente realizaron nuevas versiones del mismo.

En la actualidad, la moda "retro" ha vuelto y esto se puede ver en la tendencia a realizar juegos de diseño clásico. Unos de los ejemplos más claros que se puede ver es el Minecraft, que es un juego actual pero con gráficos pixelados que recuerdan a los Lego o juegos más clásicos. Otros como el Far Cry 3: Blood Dragon, que es un mod del juego Far Cry 3 con estética retro. Pero hay otros juegos que directamente han vuelto a la estética y diseño de los juegos retro, donde los gráficos son en 2D, como el Terraria o el Crossing Souls.

Otros campos del ocio en el que se puede apreciar esta nostalgia retro, es el del cine, con películas como Rompe Ralph, Pixels o Tron. Además hay un gran mercado de accesorios que gira entorno a todo lo relacionado con los juegos clásicos, como camisetas, llaveros, reventa de los videojuegos originales que incluso llegan a alcanzar elevados precios…. Los precios de las consolas, como Game & Watch, que pueden reproducir dichos títulos también están alcanzando unos precios desorbitados en Internet, ya que se está juntando el coleccionismo con la nostalgia.

1.3.- Estructura del trabajo

Tras esta introducción sobre la industria de los videojuegos en España, la memoria continúa organizándose como describimos a continuación.

En el segundo capítulo, se analiza el estado del arte, donde se estudian juegos de perspectiva isométrica que han tenido su propia reconstrucción, entre ellos, *La Abadía*

del Crimen, que es el juego sobre el que se va a centrar este trabajo. Además, se hará una breve introducción a las distintas herramientas de desarrollo de videojuegos, especialmente sobre aquellas que generan escenarios en perspectiva isométricas. Entre ellas se encuentra:

- Herramientas de desarrollo de ámbito profesional, como Isogenic Game Engine (Irrelon Software, 2013) o Flexible Isometric Free Engine (FIFE Team, 2013)
- Herramientas de desarrollo independiente, como Nimrod (Nimrod, 2008) o *IsoUnity* (Pérez Colado & Pérez Colado, 2014).

Asimismo, se hará referencia a las metodologías más utilizadas a la hora de producir un videojuego.

En el tercer capítulo, se concretan los objetivos y metas que se van a tratar de alcanzar a lo largo de este proyecto, para que se comprendan las motivaciones que nos han llevado a realizarlo. También se expondrá el plan de trabajo que se seguirá a lo largo de la elaboración del TFG y cómo va a ser el proceso de maduración de *IsoUnity* en el futuro, el cual se ha visto beneficiado durante la elaboración de nuestro proyecto y las investigaciones sobre optimización realizadas así como por otras tareas y requisitos realizados.

En el cuarto capítulo, se planifica el trabajo de reconstrucción que sería necesario realizar sobre el título que sirve de inspiración para este trabajo, *La Abadía del Crimen*, juego desarrollado por la compañía de videojuegos Opera Soft, se hará un estudio de cómo se estructura el juego, cuál es su jugabilidad, las vistas de las cámara, las características que tiene y el comportamiento de los personajes que dan vida al juego, explicando cómo se rediseña y se da una nueva visión al conjunto de todos estos elementos

En el quinto capítulo, se explica el uso de la herramienta *IsoUnity* para poder desarrollar *IsoAbbey* y cómo se ha extendido para poder adaptarla a los propósitos requeridos. Es reseñable el estudio sobre la lógica de los personajes y la mejor manera de implementarla en el videojuego. Por último, en este capítulo se muestra cómo funciona el control del juego para las nuevas plataformas, tanto las que permiten el uso de ratón como para interfaces táctiles.

El sexto capítulo trata sobre los inconvenientes que hemos se han encontrado al utilizar el plugin *IsoUnity* y como éste se desenvuelve al ser enfrentado ante un entorno de producción de un videojuego. Este capítulo también muestra un estudio sobre el rendimiento de la herramienta.

Finalmente en el capítulo 7 se presentan las conclusiones del trabajo, las cuales son favorables aunque de cara a una futura producción, se dan una serie de consejos a tener presentes.

Capítulo 2.- La video-aventura isométrico clásico y la producción de videojuegos

Este proyecto se centra en el desarrollo de un videojuego clásico utilizando herramientas de desarrollo actuales. Se dividirá el estado del arte en cuatro partes:

- Video-aventuras isométricas
- *La Abadía del Crimen*
- Herramientas de desarrollo
- Elaboración de un videojuego

Estas cuatro categorías engloban los temas necesarios para entender sobre qué trata éste proyecto y porque se ha realizado. Se hablará de videojuegos clásicos versionados. Y a continuación se pasará al videojuego que se tomará como objetivo para hacer una reconstrucción del mismo: *La Abadía del Crimen*. Tras esto, se dan una serie de ejemplos de versiones previas de dicho videojuego. Por último se hace un recorrido sobre ciertas herramientas que permiten la creación de videojuegos con una estética similar a éste.

2.1.- Video-aventuras isométricas

El proyecto consiste en la creación de la reconstrucción del videojuego *La Abadía del Crimen*, el cual está desarrollado bajo una perspectiva isométrica, pero utilizando técnicas actuales y tratando de ser fieles a la esencia del juego original. Llevar a cabo un juego isométrico con las técnicas de desarrollo actuales (los motores y el hardware actual manejan gráficos tridimensionales basados en polígonos en lugar de los antiguos sprites) implica una serie de retos, ya que hay que forzar la forma de trabajar de estos motores. Por ejemplo para la simulación de sprites en un motor 3d se crean quads o planos con 2 triángulos en forma rectangular y se les aplica una textura que representa el mapa de bits del sprite antiguo. A veces se agrupan sprites en la misma textura para tener un atlas o spritesheet con el que reducir las transferencias de memoria al cambiar de imagen. Esto ocurre sobre todo en las animaciones.

2.1.1.- Gráficos isométricos

La proyección isométrica es un método gráfico de representación que consiste en mostrar un objeto tridimensional plasmándolo en dos dimensiones, de manera que los tres ejes ortogonales principales (X, Y, Z), al proyectarse, forman ángulos de 120º y las dimensiones paralelas a dichos ejes se miden en una misma escala (véase la Figura 5).

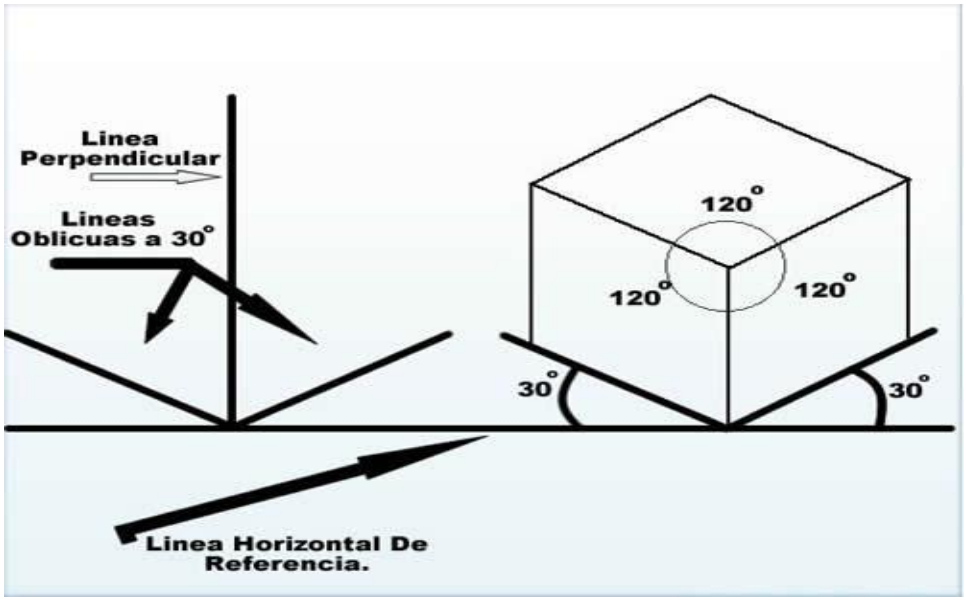


Figura 5Visión Isométrica respecto al plano y ángulos que hay que generar.

Los ejes de la X y de la Y se sitúan a 30º de la línea horizontal, pues son los que corresponden al plano horizontal. El eje Z se sitúa perpendicular a la línea del horizonte, formando ángulos de 60º con los anteriores. De este modo se verá la cara superior de un cubo con las aristas más próximas al observador formando ángulos de 120º.

El nombre de la perspectiva isométrica, proviene del griego y significa "igual medida". Esto es debido a que la escala de medición es la misma a lo largo de cada eje, cosa que no sucede en otras perspectivas. Esta perspectiva tiene el inconveniente de no mostrar la profundidad de los objetos que representa, debido a que las líneas de sus dimensiones son paralelas y los objetos no ven reducido su tamaño con la distancia.

El uso de esta técnica en videojuegos fue iniciada por Sega y Gottlieb con el juego Zaxxon. Pero más tarde Sandy White y Quicksilva fueron considerados como los precursores del uso de los gráficos isométricos con su juego Ant Attack (véase la Figura 6). En este videojuego se emuló un verdadero mundo 3D ya que aportaba al jugador una libertad de movimiento sobre los cuatro puntos cardinales (norte, sur, este y oeste) que no ofrecían los juegos de Sega.

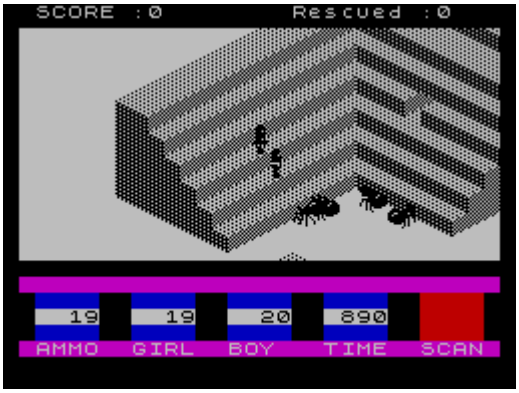


Figura 6 Ant Attack

Aunque *Ant Attack* fuera un juego que revolucionó el mercado en su día, tenía unos gráficos bastante toscos y poco atractivos. Posteriormente, con la aparición del nuevo motor gráfico de tipo isométrico por parte de Ultimate Play The Game (Posteriormente renombrada como RareWare) denominado Filmation Engine (Filmation, 1984) se pudieron desarrollar juegos como *Knight Lore*, además de tener un aspecto gráfico más atractivo, también permitía que el personaje pudiera mover objetos y explorar estancias. Con este cambio, se podían añadir puzles a los distintos niveles del juego.

2.1.2.- Historia de los juegos isométricos

En los comienzos de los 90, la perspectiva isométrica se estableció como una alternativa de visualización bastante habitual. Aparecieron aventuras de plataformas como Solstice para NES, juegos de conducción para todos los gustos, beat'em ups como el arcade Moonwalker, deportes como los primeros FIFA que permitían elegir entre vista cenital inclinada e isométrica. Otro género tradicional que también se sumó a la perspectiva isométrica fueron las plataformas como es el caso de Sonic 3D de Sega Mega Drive y algunas rebuscadas adaptaciones tridimensionales del clásico *Pac-Man* (véase la Figura 7).

Algunos géneros no se adaptaron a la perspectiva isométrica tan bien. En algunos casos servía sólo como elemento decorativo, por lo peculiar de su representación de la profundidad, sin contribuir demasiado a la parte jugable. Por ejemplo se tiene el caso de los juegos de "plataformas" como el más evidente, ya que en la actualidad se siguen utilizando con éxito las dos dimensiones para este tipo de juegos. Pero hubo otros géneros para los que, dadas sus características, esta tecnología se adaptó perfectamente como es el caso de los juegos de rol o los de estrategia como *Sim City*.

Figura 7 Juegos isométricos en los 90 y adaptaciones

Gracias a una apreciable mejora en potencia de los procesadores, a lo largo los años 90 se comenzó a usar de una manera habitual representaciones en 3D más complejas y fidedignas. Esto originó el uso de motores de perspectiva cónica, donde se proyecta un objeto 3D sobre un plano 2D, permitiendo simular de manera más realista la forma en la que se perciben los objetos. Con esta nueva forma de representación, el número de juegos isométricos descendió drásticamente, aunque en aquella época todavía se podían ver juegos isométricos de gran calidad como Baldur's Gate, donde la complejidad del mundo que se pretendía representar era demasiado grande como para poder hacerlo usando un motor 3D cónico.

Figura 8 Videojuegos isométricos de estrategia y RPG

Afortunadamente, en la actualidad se han vuelto a crear juegos con perspectiva isométrica y así se rescata, con cierto matiz nostálgico, esta vieja técnica. Esto se puede ver en videojuegos como *Game Dev Story* para iPhone o el *Lock's Quest* (5th Cell, 2008) de Nintendo DS.

2.1.3.- Ejemplos

A modo de ejemplo, se va a proceder a enumerar algunas reconstrucciones actuales de clásicos isométricos, describiendo qué elementos se han mantenido y cuales se han modificado, para posteriormente adaptarlos a las técnicas de desarrollo y de diseño actuales. Y así poder utilizar esta información y tratar de aplicarla a *IsoAbbey*.

Knight Lore

JUEGO ORIGINAL

Fue un juego que revolucionó los 8 bits. En el año 1984, los desarrolladores del Knight Lore, los hermanos Stamper pertenecientes al equipo Ultimate, presentaron al mundo Filmation, que permitía la creación de entornos de visión 3D, ideal para aventuras de arcade basadas en plataformas como *La Abadía del Crimen*. Este motor resulta bastante interesante ya que comparte muchas características que se utilizan en. Por ejemplo, los jugadores pueden moverse con el personaje en cuatro direcciones, al igual que Guillermo (el principal personaje jugable y protagonista del juego). Además se pueden introducir objetos en el entorno, característica también presente en *La Abadía del Crimen*, que es fundamental para un juego de aventura. Esta técnica sería igualada, e incluso superada en videojuegos posteriores.

EL REMAKE

Se pueden destacar dos reconstrucciones del videojuego Knight Lore. La primera reconstrucción que fue desarrollado por *Devilish Games* en el 2002 (véase la Figura 9), que destacar por las siguientes características aportadas:

- Color en los personajes y escenarios.
- Más definición en los niveles, mejorando con ello la resolución del juego.
- Una mejora en las texturas de las paredes del castillo, las cuales antes no se podían generar, además de una serie de efectos en los objetos, como el fuego de las antorchas.
- Sonidos de alta calidad.
- Variación de la velocidad del juego, consiguiendo simular la jugabilidad del juego original, en ciertos aspectos con una gran cantidad de elementos en pantalla el juego se ralentizaba.
-

El único inconveniente que tenía este remake es que los bloques desaparecían algunas veces al pisarlos.

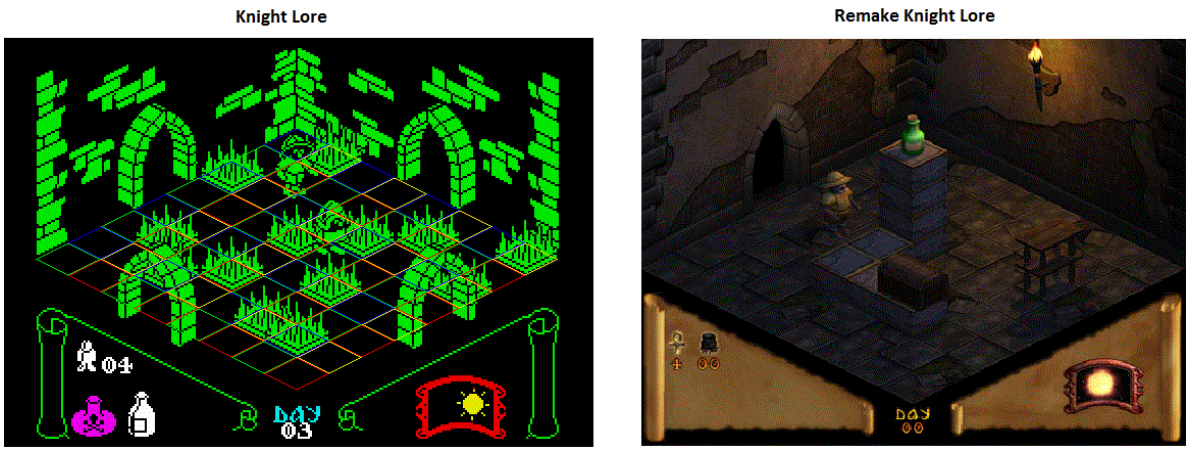


Figura 9 Contraste de imágenes entre el juego original del Knight Lore y su Remake

El siguiente remake del que se hablará es el Ricardo Galvany (Galvany, 2006). Donde lleva el juego original a un entorno 3D. Para desarrollar el remake del knight Lore no parte desde cero como es habitual, sino que aprovecha al máximo el juego original. Esto implica que se vea forzado a trabajar sobre una par de emuladores:

- **ZX-Spin:** lo utiliza para el proceso de ingeniería inversa. Este proceso tiene como finalidad la de desviar el flujo de datos en el momento de generar las imágenes y así a partir de algoritmos 3D propios, se crea la escena a partir de los datos generados por el propio juego. Esto además hace que la jugabilidad original quede intacta.
- **Aspectrum:** lo utiliza como base del proyecto, ya que dispone de su código fuente y su compatibilidad con muchas plataformas.

Como compilador para el remake utilizó el MinGw con la biblioteca Allegro. Y finalmente para la representación gráfica 3D utilizó la biblioteca OpenGL.

De esta reconstrucción se puede destacar el minucioso trabajo en la extracción de las texturas y sprites a través de ingeniería inversa. También el pasar los fondos, objetos y personaje de una baja resolución en 3D a una alta resolución. Para ello tuvo que hacer uso del documento de *Knight Lore data format* de Chistopher Jon Wild. Donde explica que a través de la utilización de un sistema de ecuaciones (véase la Figura 10), se puede transformar los objetos 3D low a 3D high y viceversa.

$$X_H = (X_L \cdot 16) + (X_1 \cdot 8) + 72$$
$$Y_H = (Y_L \cdot 16) + (Y_1 \cdot 8) + 72$$
$$Z_H = (Z_L \cdot 12) + (Z_1 \cdot 4) + ScreenZ$$

Figura 10 Sistema de ecuaciones de Christopher Jon definidas en el documento Knight Lore data format

Por último es interesante mencionar un problema que surgió, al permitir la coexistencia entre el emulador con el juego original y la del entorno 3D (véase la Figura 11). Este problema se encuentra a la hora de pasar el entorno a 3D. Ya que OpenGL es una máquina de estados, y una vez se le ha pasado el control, itera indefinidamente hasta que se termina el programa o se le obliga a terminar. Haciendo que el emulador se

quedara parado al ejecutar el entorno 3D. Para solucionar este problema se cambió el planteamiento de las modificaciones del emulador. En lugar de ejecutar la función que realiza una iteración del emulador, y luego ejecutar la que genera el entorno 3D, se optó por conceder el control absoluto al entorno 3D y mediante funciones propias de OpenGL, que permiten especificar qué hacer en periodos de inactividad, ir concediendo iteraciones a la función de emulación.

Figura 11 Diferentes vistas de una misma habitación del Knight Lore 2006 y el original

Batman

JUEGO ORIGINAL

John Ritman y Bernie Drummond fueron otros pioneros de los juegos de los 80, llevando la perspectiva de los juegos isométricos más lejos que Ultimate. Dejaron atrás los videojuegos de deportes que gozaron de un gran éxito, para desarrollar los videojuegos de perspectiva isométrica *Batman* y *Head Over Heels*.

EL REMAKE

El remake, uno de los pioneros en la escena de la reconstrucción, fue creado por Kakarot en el año 2000 para PC. Con la ayuda de un desarrollador gráfico, supo recoger la esencia del juego original, manteniéndose fiel al mismo, a la vez que le otorgaba la modernidad necesaria, gracias a los adelantos de las nuevas tecnologías (véase la Figura 12).

Posteriormente, el juego fue pasado a Game Boy Advance, sin perder ni un ápice de la calidad original.

Figura 12 Contraste de imágenes entre el juego original Batman y su Remake.

Head Over Heels

JUEGO ORIGINAL

Ultimate empezó el desarrolló Head Over Heels pero luego fue puesto en manos de Jon Ritman y Bernie Drummond quienes siguieron con su desarrollo hasta perfeccionarlo, llevando al motor Filmation hasta su máxima expresión.

EL REMAKE

Los autores de esta reconstrucción presentan un trabajo muy bien realizado del juego de Ocean, siendo lo más fiel posible al título original. Las mejoras que se pueden apreciar son (véase la Figura 13).

- Resolución
- Colores
- Sonido

Además crearon una herramienta para desarrollar videojuegos de este estilo: Filmation, de la cual se entrará en detalle más adelante.

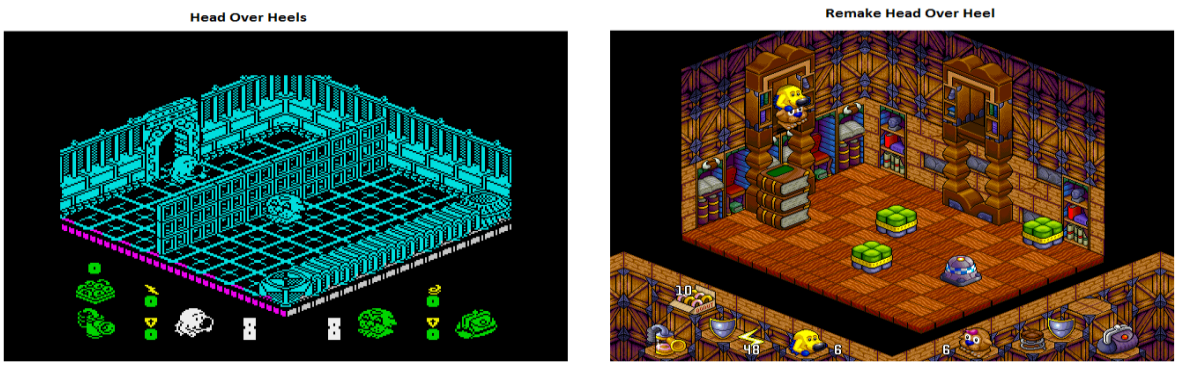


Figura 13 Contraste de imágenes entre el juego original Head Over Heels y su Remake

Alien 8

JUEGO ORIGINAL

En 1984, Ultimate sorprendió con el videojuego Knight Lore. Un año más tarde, aprovechando el mismo motor, los hermanos Stamper nos trasladaron a una nave espacial en problemas. El jugador tomaría el control de un robot de mantenimiento que debe reactivar las cámaras criogénicas de la nave antes de que los astronautas sufrieran daños irreversibles.

La mecánica era la misma que en Knight Lore, sólo que con ligeras modificaciones; ahora se tenían que colocar los objetos específicos en distintos lugares. En algunos momentos del juego se puede guiar a un robot zapador para limpiar nuestro camino de minas.

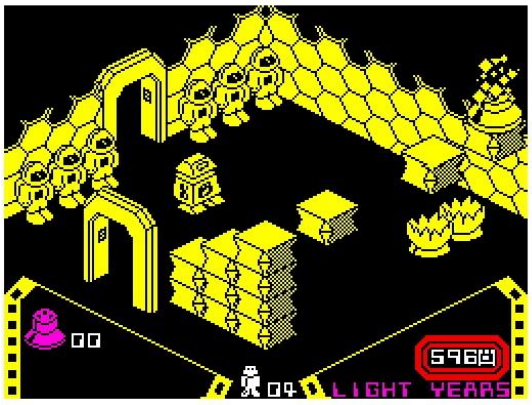
Pocos cambios hubo de Knight Lore a Alien 8, se tuvo que esperar a que saliera más adelante el motor Filmation.

EL REMAKE

Veintitrés años después se desarrolló una reconstrucción de Alien 8, donde destaca el renovado menú de opciones, presidido por un ordenador Enterprise y amenizado con la música original perfectamente adaptada. Es importante destacar que esta versión permitía la opción de redefinir ciertas teclas y la modificación de varios aspectos del juego como el volumen de la música, diversos efectos o el idioma.

Una vez dentro de la nave, se puede observar que el trabajo de modernización ha sido exhaustivo (Véase la Figura 14). El motor del programa fue desarrollado por Ignacio Pérez Gil, que es prácticamente idéntico al original. El apartado gráfico corresponde a David Olías, que tenían gran calidad, pero al mismo tiempo esta gran calidad, distancia la estética del juego original al remake. Por último, los autores han introducido una nueva y divertida secuencia animada, distinta al original, que se puede ver cuando el jugador perdía todos sus puntos de vida.

Alien 8



Remake Alien 8

Figura 14 Contraste de imágenes entre el juego original Alien 8 y su Remake

2.2.- La Abadía del Crimen

La Abadía del Crimen es un videojuego del género aventura desarrollado por la compañía Opera Soft en 1987. Sus desarrolladores fueron, el programador Francisco Menéndez y el diseñador gráfico Juan Delcán. Este juego está considerado como uno de los mejores dentro de la llamada época, "la edad de oro del software español", comprendida entre 1983 y 1992 (Esteve, 2014),

El videojuego se ambienta en la novela *El Nombre de la Rosa* (Eco, 1980). La novela se sitúa en Italia en el año 1327 y narra el viaje de Guillermo de Baskerville y su discípulo Adso de Melk a una abadía, para organizar una futura reunión del papa con los líderes franciscanos. La trama se complica cuando una serie de asesinatos, inspirados según el bibliotecario Jorge de Burgos en unos pasajes del Apocalipsis, esto sacude a la abadía y será Guillermo el encargado de resolver el misterio. Hay una película de título similar que empieza con la llegada de Guillermo a la abadía y que sigue la trama del libro. El videojuego, al igual que la película *El Nombre de la Rosa* sigue la historia de la novela.

Este juego atrajo nuestra atención por su diseño de los gráficos en 8 bits y su perspectiva isométrica, además de ser un juego muy bien considerado cuando salió a la venta. El videojuego se desarrolló para distintas plataformas como: Amstrad CPC, Spectrum, MSX y PC. Y dependiendo de la plataforma se comercializó en diferentes formatos como el cassette o CD-ROM.

2.2.1.- Creadores de La Abadía del Crimen

Francisco Menéndez:

Ingeniero de telecomunicaciones por la universidad politécnica de Madrid. Era el programador del videojuego. Trabajo en varias empresas españolas de desarrollo de videojuegos como Indescom, Made in Spain y Opera Soft. Menéndez se decidió a desarrollar este videojuego tras leerse el libro de *El nombre de la rosa* y darse cuenta de la historia tan compleja que tenía detrás sería un buen argumento para un juego.

El desarrollo del juego duró más de un año. Tras finalizarlo, a la hora de poner el nombre al videojuego deicidio pedir permiso a Umberto Eco, para poder llamarlo como el libro. Pero este se negó porque interpreto mal el propósito de la utilización del nombre de la obra para el videojuego.

Debido a numerosos problemas con la distribución del videojuego, Menéndez decidió separarse de la programación de los videojuegos y comenzar con un nuevo proyecto, que denominó Memoria matricial inteligente. La cual consistía en almacenar datos y que al mismo tiempo se pudiesen ejecutar instrucciones, lo que supondría una paralelización a bajo coste. Pero en el año 1999 en Sevilla sucede lo peor, Menéndez decide quitarse la vida arrojándose desde su piso. Este acontecimiento está supuestamente relacionada a la excesiva presión a la que estaba sometido debido a la gran inversión que había acometido en su proyecto. Un trágico final que según la hermana de Francisco, Malena Menéndez, no es cierto del todo.

Frases destacadas de Francisco Menéndez:
- Prefiero el reconocimiento de la gente al dinero.

Juan Delcán:

Fue el encargado de elaborar el entorno gráfico de *La Abadía del Crimen* mientras estaba estudiando la carrera de arquitectura en Madrid. Su trabajo fue muy complejo, esto era debido a los escasos recursos que había en la época de los 80 para poder crear entornos gráficos. Tuvo la genial idea de diseñar escenarios isométricos con diferentes planos que en aquella época no era común. También diseñó la portada de este juego, al igual que del videojuego *Sir Fred*. Tras terminar con el proyecto de la abadía salió del mundo de los videojuegos, al igual que Francisco.

2.2.2.- Versiones de La Abadía del Crimen

A partir de la salida de *La Abadía del Crimen*, aparecieron un gran número de seguidores. Estos afirman que es el mejor juego de ordenador hecho en España, debido a las nuevas características, como el sistema de enfoque de la cámara sobre la abadía. Algunos de estos seguidores decidieron hacer reconstrucciones del juego donde intentarían mejorar ciertos aspectos del juego. Ya que dispondrán de herramientas más modernas y no como las que tuvieron Menéndez y Delcán en su día.

La primera reconstrucción que se hizo fue la de Antonio Giner en 1999, que lo implementó en la plataforma Win32 y MSDOS. Mejoró el entorno gráfico dándole colores al juego y además dio la posibilidad de guardarlo, algo que originalmente no se podía hacer, permitiendo terminar el proyecto de una forma más fácil. (Véase la Figura 15).

Figura 15 Remake de La Abadía del Crimen de Antonio Giner

Otro remake que se hizo en MSX2 y Java fue el de Manuel Pazos y Armando Pérez en el 2001. El primero de ellos creó una versión parcheada de la original de MSX 2, en la cual se pueden grabar partidas. Partiendo de ella, Manuel Pazos realizó una extraordinaria versión en 16 colores, también para MSX 2 y Turbo R, obteniendo como resultado tres paletas de colores diferentes según el momento del día, y el sonido original del MSX (véase la Figura 16).

Figura 16 Remake de la Abadía del Crimen de Manuel Pazos y Armando Pérez

La Abadía del Crimen también fue adaptada al 3D con el juego *The Abbey* en el 2008 (véase la Figura 17). Pero más que un remake fue como una conmemoración del juego de *La Abadía del Crimen*, ya que no se parecía en muchos aspectos al juego original, como por ejemplo en cuanto a jugabilidad.

Figura 17 Remake 3D de La Abadía del Crimen llamada The Abbey

La última reconstrucción es *La Abadía del Crimen Extensum*. La cual está desarrollado en Java, haciendo que sea compatible con los sistemas operativos Windows, Linux y Mac OS X. Además se pretende que el juego sea un ejecutable freeware, es decir, que se pueda descargar y ejecutar de forma completamente gratuita.

El juego fue presentado en RetroBarcelona 2014. Celemín y Pazos desvelaron el primer tráiler de esta nuevo reconstrucción en el que llevan trabajando meses después de la insistencia de Celemín.

Celemín, un apasionado de *La Abadía del Crimen*, había puesto en marcha un par de remakes que no fue capaz de acabar. Insistió a Manuel Pazos para que le ayudara con el desarrollo y creará un nuevo motor para comenzar sobre la base del juego original.

Uno de los objetivos de *La Abadía del Crimen Extensum* es claro: ofrecer una experiencia que vaya más allá de los dos remakes lanzados en su día por Manuel Pazos y Antonio Giner. "Hay más personajes y la trama intenta ser más fiel al libro aunque nos hemos encontrado con algún problema porque el guion del juego original era lineal y en el remake tenemos que controlar la narrativa porque el jugador tiene más libertad de movimientos"(Celemin, IGN, 2 de diciembre de 2014).

El objetivo de Pazos y Celemín es que el juego sea más accesible que la obra original de Menéndez y Delcán publicada por Opera Soft. Por ello introdujeron un sistema de pistas, por ejemplo: que el Abad diga tras la bienvenida: “Ubertino llegó hace unos días. Le he visto en la iglesia hace un momento”. Así, el jugador sabe que si va a la iglesia se encontrará con Ubertino, pero no es algo a lo que estemos obligados hacer. Al igual que Ubertino, Guillermo y Adso también darán pistas como: “Sígueme Adso, creo que es hora de conocer el scriptorium' o “Maestro, oigo pasos. Alguien ha salido de la celda de al lado”.

Además añadieron escenas cinemáticas donde todo ocurre de forma automática, como en la autopsia de Venancio (véase la Figura 18) que sirven para explicar parte de la trama sin necesidad de la intervención del jugador. Quisieron que el juego fuese más sencillo que el original y evitara situaciones en las que el jugador no supiera qué hacer.

Otro de los elementos que han incorporado en este nuevo remake es la mejora en la resolución, que permite adaptar el juego a una resolución de 16:9, pudiendo ver el juego en pantallas de mayor tamaño.

Por último diseñaron una abadía más grande, lo que ha implicado que se retoquen los horarios, ya que en el videojuego original obligan al protagonista a ir de una punta otra de la abadía en un margen de horario establecido, ya fuese para ir a misa como a comer. Esto impedía que el jugador dispusiera de la libertad de acción de recorrer la abadía, haciendo el juego original demasiado lineal. Esta mejora está contemplada en nuestro estudio.

Figura 18 Remake de La Abadía del Crimen de Manuel Pazos y Daniel Celemín

2.3.- Herramientas de desarrollo

Las herramientas de desarrollo de videojuegos nos permiten desarrollar diferentes juegos, pero hay que tener un conocimiento amplio sobre programación, esto es debido a la complejidad que requieren para adaptarlas a ciertos videojuegos. Estas herramientas se caracterizan por poder usar la misma infraestructura en distintos juegos y portar la misma implementación de un juego a distintas plataformas.

Los motores de videojuegos suelen compartir unas funcionalidades típicas. Estas son Motor gráfico (renderizado de gráficos 2D/3D), Motor físico (p.ej. detección de colisiones), Lenguaje de scripting, sonido, animaciones, redes, inteligencia artificial, etc.

Dentro de las alternativas que hay en las herramientas de desarrollo, aparecen cuatro grandes grupos:

1. Desarrollo nativo usando el SDK estándar para una plataforma (junto con bibliotecas para videojuegos). Ejemplo: Android, HTML5 & JavaScript.
2. Bibliotecas multiplataforma (para lenguajes de propósito general). Ejemplo:OpenGL (gráficos), OpenAL (audio), etc.
3. Lenguajes para el desarrollo de videojuegos. Ejemplo: Lua (World of WarCraft, Angry Birds).
4. IDE’s para el desarrollo de videojuegos. Ejemplo: GameMaker: Studio.

A continuación se procede a mencionar una serie de herramientas muy utilizadas en la actualidad para el desarrollo de videojuego. Estas herramientas tienen que estar en constante evolución, ya que cada vez el mercado exige más calidad y hay más competencia.

2.3.1.- Unreal Engine

Es un motor de juego de PC y consolas creados por la compañía Epic Games. Implementado inicialmente en el shooter en primera persona *Unreal* en 1998. Es uno de los motores más usados en la industria y ha sido usado en incontables videojuegos como: *Batman: Arkham Asylum, Gears of War, Kingdom Hearts III o Shenmue III.* También ha sido utilizado en otros géneros como el rol y en juegos de tercera persona. La última versión de Unreal Engine utiliza como código fuente C++, siendo compatible con varias plataformas como PC (Microsoft Windows, GNU/Linux), Apple Macintosh (Mac OS, Mac OS X) y la mayoría de consolas. Unreal Engine también ofrece varias herramientas adicionales de gran ayuda para diseñadores y artistas.

2.3.2.- CryEngine

CryEngine es un motor de juego creado por la empresa alemana desarrolladora de software Crytek. Originalmente un motor de demostración para la empresa Nvidia (Nvidia, 1993), que al mostrar un gran potencial se implementa por primera vez en el videojuego *Far Cry,* desarrollado por la misma empresa creadora del motor. CryEngine tiene la versión 2 y 3 que se utilizan para desarrollar juegos como la saga *Crysis.*

2.3.3.- RPG Maker

Serie de programas para el desarrollo de videojuegos de rol (RPGs) creada por ASCII Corporation, parte de la corporación Enterbrain. Permite al usuario crear sus propios videojuegos de rol, además, el engine incluye un editor de mapas, un editor de eventos y un editor de combates.

Todas las versiones de RPG Maker de PC tienen el RTP (Run Time Package) que incluye materiales como gráficos para mapeados, personajes, música, efectos de sonido, materiales gráficos y sonoros personalizados, etc. que pueden ser utilizados para crear nuevos juegos.

2.3.4.- Filmation Game Creator

Esta herramienta fue utilizada para crear las reconstrucciones de Batman y el Head Over Heels. Sirve para crear juegos isométricos de estética Filmation de una forma intuitiva y ha sido desarrollado por Hugo Gil (véase la Figura 20).

Estéticamente es fiel a los juegos originales de Filmation. Con este editor, se pueden crear habitaciones, usar nuevos sprites, añadir sonidos y decorados, animaciones,

movimientos a los objetos y crear paquetes para distribuirlos libremente por tu juego. Además de contener una gran cantidad de sprites de los juegos originales.

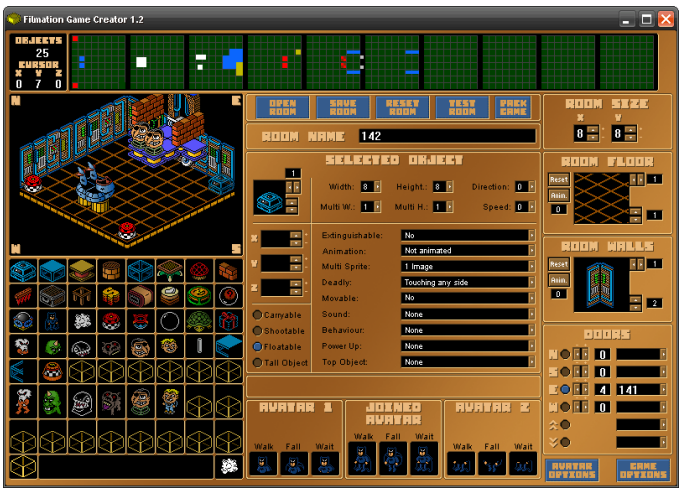


Figura 19 Captura de la herramienta de Filmation Game Creator

2.3.5.- Nimrod

Es otra herramienta para crear videojuegos con perspectiva isométrica (véase la Figura 21). La interfaz del programa está hecha con Photoshop y luego procesado con C++ y OpenGL. Es posible remover objetos del mapa establecer franjas horarias y con ello añadir sombras y luces según la hora del día. Pero la herramienta aún se encuentra en desarrollo, ya que es un proyecto de un desarrollador independiente con seudónimo "herr-o".

Esta herramienta ha sido utilizada por el estudio de desarrollo de videojuegos OddGames en la creación de Medieval History.

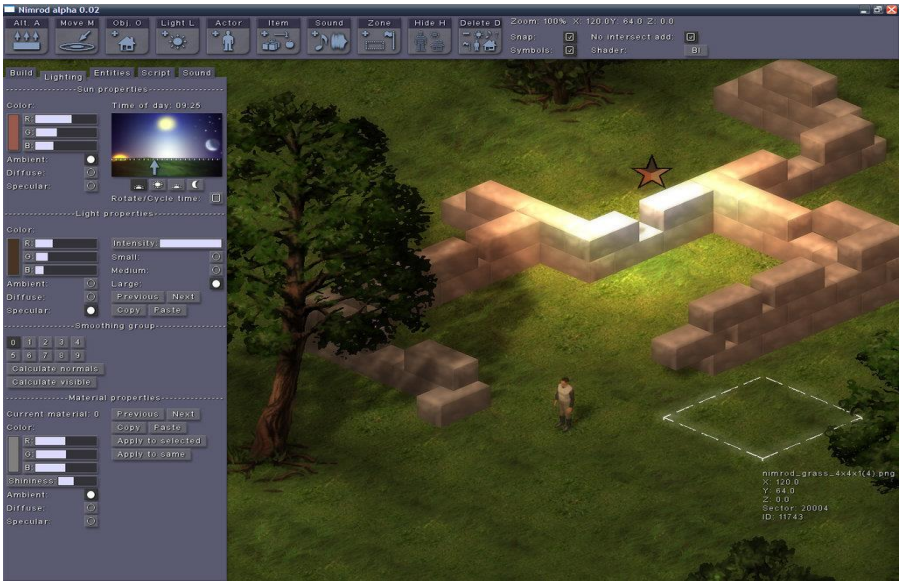


Figura 20 Captura de la herramienta Nimrod

2.3.6.- Isogenic Game Engine

Es una herramienta de desarrollo de videojuegos que utiliza las aplicaciones web para crear videojuegos. El lenguaje de desarrollo que utiliza es HTML5 y Javascript. Los mapas son construidos a base de baldosas 2D y tiene soporte de almacenamiento con Node.js, además de tener soporte para MongoDB y MySQL. Motor isométrico que aún está en desarrollo y se lanzará una versión beta pronto (véase la Figura 22).

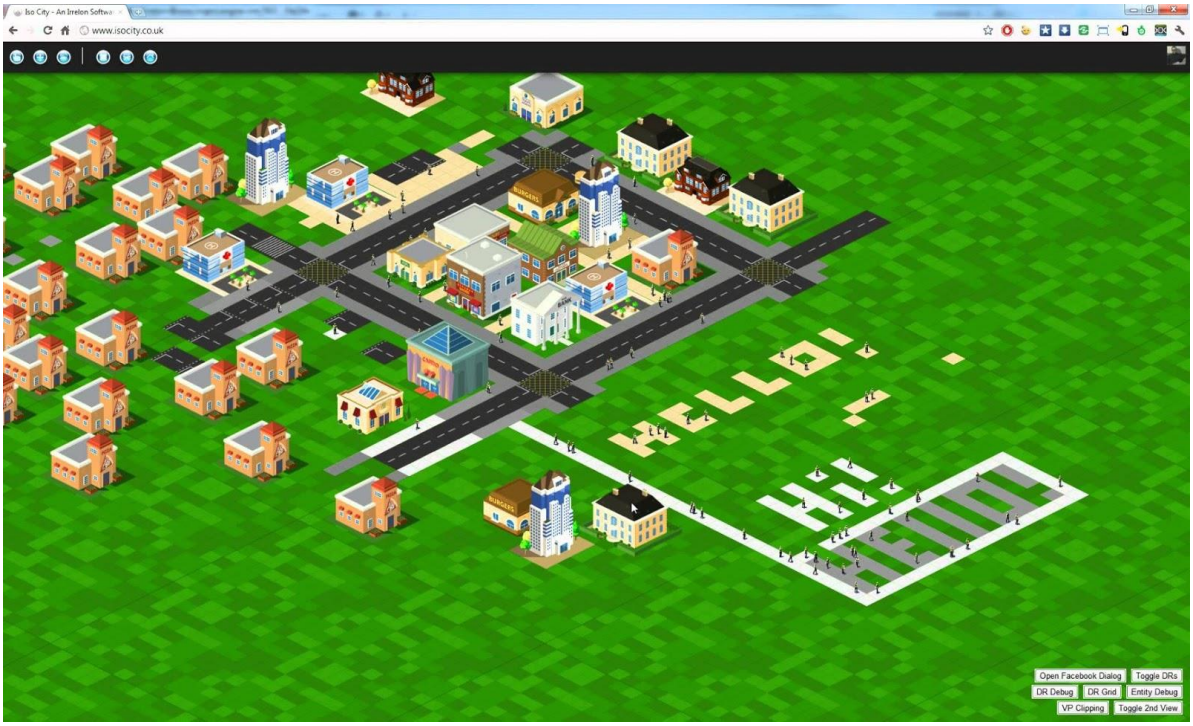


Figura 21 Captura de la herramienta Isogenic Game Engine

2.3.7.- Flexible Isometric Free Engine

Flexible Isometric Free Engine (FIFE) es una herramienta de código abierto, con motor de juego multiplataforma. El lenguaje de desarrollo que utiliza es C ++, además de ser compatible con el scripting Python. Está registrado bajo la licencia Creative Commons de GNU (LGPL), que permite la creación de juegos comerciales independientes (véase la Figura 23)

Los desarrolladores de esta herramienta se inspiraron en la perspectiva isométrica que se encuentra en el Fallout. No obstante, se puede utilizar para crear cualquier juego 2D. El enfoque no tridimensional asegura que juegos desarrollados con FIFE, puedan funcionar en hardwares de pocas prestaciones, y también simplifica la programación y el marco de la creación de contenidos. El motor en sí es un intermediario entre los clientes y las bibliotecas externas. Un ejemplo de un cliente sería un juego construido en la parte superior del motor, y OpenAL u OpenGL sería dos bibliotecas externas. Se divide en las extensiones del núcleo y motor de rendimiento crítico (implementadas con Python), que están unidas entre sí a través de envoltorios SWIG.

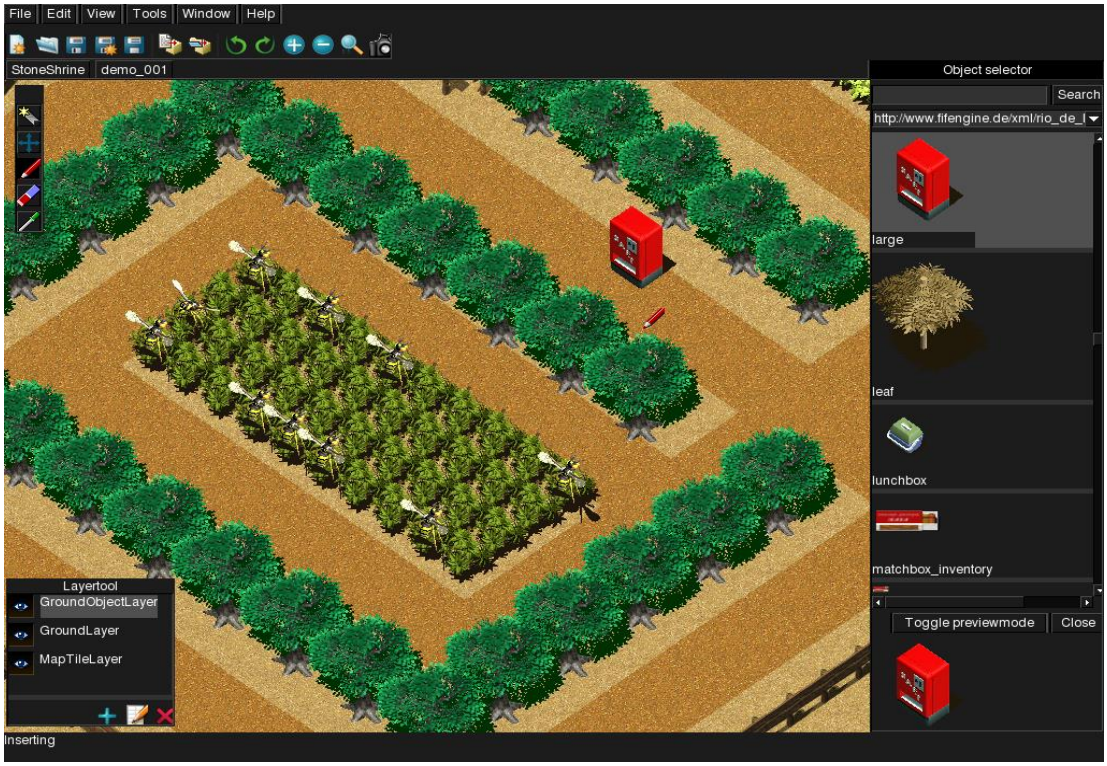


Figura 22 Captura de la herramienta Flexible Isometric Free Engine

2.3.8.- Unity

Es un motor de videojuegos multiplataforma creado por Unity Technologies. Con el cual se han desarrollado juegos como *RollerCoaster Tycoon World* o el *Wasteland 2*. Unity está disponible como plataforma de desarrollo para Microsoft Windows y OS X, permitiendo portar un juego hasta a 21 plataformas diferentes con un solo código, por lo que los desarrolladores pueden crear una versión de su juego para diferentes plataformas, incluso Oculus Rift. Gracias al *plugin* web de Unity, también se pueden desarrollar videojuegos de navegador para Windows, Mac y Linux.
Desde el sitio web oficial se pueden descargar dos versiones: Unity Personal Edition y Unity Professional Edition. La primera versión ofrece el Motor con todas las prestaciones, sin reglas y desarrollo en todas las plataformas pero con restricciones. Mientras que en la versión Profesional se encuentran las mismas características de la Personal Edition más la opción de personalizar la pantalla de inicio. Además de estas dos versiones se pueden añadir a Unity extensiones tanto gratuitas como de pago llamadas asset. Estos asset se pueden encontrar en la tienda de unity y en su página web. Las asset relacionadas con perspectiva isométrica son las siguientes:

- *Isometric 2.5 toolset*. Contiene las siguientes características:
    - La clasificación automática de azulejos y objetos isométricos 2D.
    - Clasificación de objetos con tamaño de una baldosa individual, así como el tamaño de varias baldosas.
    - Funciones auxiliares para convertir coordenadas isométricas en coordenadas del editor de Unity.
    - Colocación y alineación de objetos en el editor de Unity.
    - Soporte para físicas. (Colisionadores, activación y Colisión eventos)

- *Ulimate Isometric toolkit.* Contiene las siguientes características:
  - Clasificación de profundidad de Isométricos.
  - Colisionadores de Isométricos.
  - Cuerpos Rígidos e Isométricos.
  - Extensiones editor de apoyo para acelerar el diseño de niveles, como:
    - Personalización del manejo direcciones isométricos.
    - Posicionamiento preciso.
  - Aproximadamente 480 sprites diferentes para 3 configuraciones diferentes.
    - Medieval Kingdom.
    - Edificios de ciudades/Coches/Carreteras/…
    - Exoplanet/Sci-Fi.
  - Modelo de controles para personajes para:
    - Juegos TurnBased.
    - Movimientos continuos.
    - Movimientos discretos.
    - Movimientos restrictivos por deteccion de colision.
  - 5 Modelos de escenas.
  - Funciones fáciles.
  - Mover objetos alrededor en una manera más natural...
  - Cuadrícula para los mapas.
  - Ahorra tiempo en la generación de niveles.

- *2D Isometric Tile.* Contiene las siguientes características:
  - Actualización 1.2, viene con 140 baldosas nuevas como costa, mas baldosas de transacción, nueva pared de piedra, diferentes tipos de madera, bloques de agua y muchas tipos más.
  - Actualización 1.1 hay baldosas de lava,varias baldosas de ladrillo,baldosas de nieve, revestimiento agrietado y muchos más tipos.
  - Las baldosas vienen con gran definición y pueden ser pasadas a menor resolución mientras conserva gran cantidad de detalles.

- *Isometric Shooter*.Contiene las siguientes características:
  - Nuevos predefinidos añadidos.
  - GUI sustituido por el nuevo Unity interfaz de usuario.
  - Iconos añadidos a la interfaz de usuario.
  - Clasificación de ejemplos de sonido añadidos.
  - Artículos para puntos de XP.
  - Clasificación por puntos de XP.
  - Añadido el sonido grito cuando mueres.
  - Tipo de enemigo comandante.
  - Tipo de arma escopeta.
  - Calcomanías de sangre y explosión física.
  - Soporte para teléfono móvil.
  - Misión de campaña"Wolf".
  - I.A. en el modo de comportamiento.
  - I.A. dispositivo de patrulla.
  - Explotar obstáculos.
  - La munición puede ser recargada.

- Modo campaña.
- Menú de inicio con link a Jugador.
- Botín al matar.
- Subida de nivel.
- XP contador al matar.
- Directional material change.
- Material de animación.
- Armas de balística.
- Proyectiles con área de daño.
- Camara de repetición al morir.
- Estilo cómic de disparo isométrico.
- Creación fácil de mapas avanzados con calcomanías de sombras.
- Selección de personaje y mapa.
- Entornos destructibles.
- Producir valores para daño, vida y munición.
- Jugador con movimientos estándar.
- Selección de arma con la rueda del ratón.
- Paquete de elementos botiquín y munición.
- I.A. simple.
- Banda sonora y efectos especiales.
- Eventos.

2.3.9.- *IsoUnity*

IsoUnity es un conjunto de herramientas para Unity orientado al desarrollo de videojuegos de acción-aventura y de estilo gráfico isométrico 3D.

Esta herramienta ha sido desarrollada por Iván Pérez Colado y Víctor Manuel Pérez Colado para un TFG bajo el título: Un conjunto de herramientas para Unity orientado al desarrollo de videojuegos de acción-aventura y estilo retro con gráficos isométricos 3D (año 2014). Su última versión está adaptada para usarla en Unity 5(véase la Figura 19).Las características a destacar para edición de juegos isométricos son:

- Adaptar texturas a baldosas con un programa propio de la herramienta.
- Inserción de sprites propios y animados.
- Creación de varias baldosas al mismo tiempo.
- Edición del número de cubos tanto en alto como ancho.
- Gestión de entidades:
  - Distribución a los mapas.
  - Interpretación de secuencias.
  - Utilización de emociones.
  - Manejo de variables globales.
- Comportamiento:
- Interfaces:
  - Gestión de Interfaces.
  - Interfaz de diálogo.
  - Interfaz de inventario.
  - Interfaz de selección de acciones.
  - Interfaz de botones de pantalla.
- Controles adaptados para teclado como pantalla táctil.

- Secuencias que funcionan como un conjunto de de acciones ordenadas en forma de árbol. Permiten una programación en alto nivel, sin conocimiento de scripting. Se compone de tres piezas:
    - Hablador: Se encarga de almacenar, editar y lanzar la secuencia.
    - Intérprete: Se encarga de ejecutar la secuencia en tiempo de ejecución.
    - Editor: Se encarga de modificar la secuencia en tiempo de edición.
- Utilizando las herramientas se pueden diseñar juegos de aventura teniendo bajos conocimientos de programación.
- El editor de mapas cuenta con múltiples técnicas de mapeado para que se pueda aprovechar el potencial tridimensional.

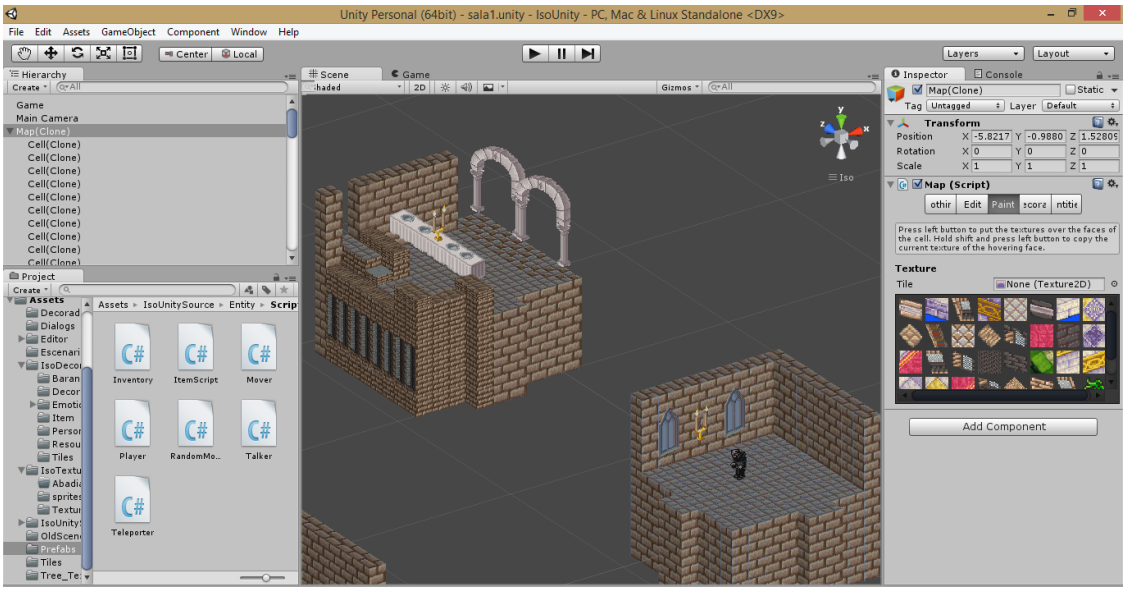


Figura 23 Captura de la herramienta IsoUnity

2.4.- Elaboración de un videojuego

La elaboración de un videojuego sigue una serie de pasos complejos y tediosos. Para poder afrontar estos pasos, es necesario reunir un grupo de profesionales cualificados en diferentes campos (jefe de proyecto, director de finanzas y administración, Productor, Director artístico, Guionista, Programador de Herramientas, Diseñador de videojuegos). La industria se caracteriza por tener poca experiencia en el desarrollo de videojuegos para los dispositivos táctiles, como los móviles y tabletas, que suelen ser desarrollados por pequeños estudios como PopCap. Uno de los retos que tiene que afrontar la industria a la hora de desarrollar un videojuego es desarrollarlo en un tiempo limitado. Por ejemplo, proyectos en plataformas web y PC se deben desarrollar entre dos y doce meses, pero los juegos tripleA pueden tardar hasta 10 años en ser terminados.

La mayoría de las metodologías utilizadas para el desarrollo de videojuegos siguen los principios ágiles, por ser iterativas e incrementales, tener interacción frecuente con el cliente y ser flexibles ante los requerimientos cambiantes. Otra característica de estas

metodologías es que las decisiones se toman en base a la experiencia, sin existir un proceso definido ni técnicas específicas a seguir. En promedio, cada proyecto lo realizan de tres a cuatro personas que cubren los roles de productor, programador, artista gráfico, diseñador de juego y artista sonoro. Las tareas de este último habitualmente son realizadas por empresas externas especializadas. La tendencia a utilizar metodologías ágiles para videojuegos tomó fuerza en los últimos años, por existir varios casos de empresas en la industria que logran adaptar estas metodologías y además ser un tema actual en uno de los eventos principales como es la Game Developer Conference (GDC) (Keith, 2009). Entre las empresas con casos de éxito documentados se encuentran Large Animal Games (Tobey, 2008), Crytek (Crytek, 2008), DICE (Nutt, 2008) y Nokia (Foe, 2008) que utilizan Scrum, Titus Interactive Studios (Demachy, 2008) que utiliza XP y High Moon Studios (Keith, 2008) que utiliza ambas. A pesar de los beneficios que reportan, ninguna de estas adaptaciones está especificada formal y públicamente.

Por ello hemos usado la metodología de desarrollo SUM, la cual busca adaptarse a la realidad del mercado actual y hacer un aporte al desarrollo de su industria. En particular se toman Scrum y XP como base de SUM por la existencia de casos de éxito y los beneficios que reportan para desarrollo de videojuegos. Además, la actual utilización de algunos de sus principios en la industria local facilita su adopción.

En la sección 2.4.1 y 2.4.2 se resumen los principales aspectos de la metodología, sus Roles y ciclo de vida. La metodología SUM para videojuegos tiene como objetivo desarrollar software de calidad en tiempo y coste, así como la mejora continua del proceso para incrementar su eficacia y eficiencia. Pretende obtener resultados predecibles, administrar eficientemente los recursos y riesgos del proyecto, y lograr una alta productividad del equipo de desarrollo. SUM fue concebida para adaptarse a equipos multidisciplinarios pequeños (de tres a siete integrantes que trabajan en un mismo lugar físico o estén distribuidos), y para proyectos cortos (menores a un año de duración) con alto grado de participación del cliente.

La definición de la metodología se basa en el Software and Systems Process Engineering Metamodel Specification (SPEM) 2.0 (Object Managment Group, 2007), un meta-modelo para describir procesos y metodología desarrollado por el Object Management Group (OMG). Una ventaja de utilizar SPEM es que su estructura permite especificar el proceso de desarrollo de videojuegos sin mencionar prácticas específicas, lo que lo hace flexible y adaptable a cada realidad. Para especificar la metodología se utiliza Eclipse Process Framework (EPF) (Eclipse Foundation, 2008) ya que provee un marco de trabajo extensible basado en los conceptos de SPEM 2.0 para definir y manejar procesos de desarrollo de software.SUM adapta para videojuegos la estructura y roles de Scrum descrita por Ken Schwaber (Schwaber & Beedle, 2001). Se utiliza esta metodología ya que brinda flexibilidad para definir el ciclo de vida y puede ser combinado fácilmente con otras metodologías para adaptarse a distintas realidades.

2.4.1.- Roles

Las metodologías ágiles están definidas por 4 roles:

1. Equipo de desarrollo.
2. Productor interno.
3. Cliente.
4. Verificador beta.

El equipo de desarrollo tiene las características del Scrum team, pero a diferencia de Scrum, se definen sub-roles dentro del equipo. Estos se corresponden con los que se utilizan habitualmente en la industria local y son los de programador, artista gráfico, artista sonoro y diseñador de juego. Es necesaria esta definición ya que se requiere una alta especialización para satisfacer las distintas disciplinas que involucra el desarrollo de videojuegos, aspecto no contemplado en Scrum.

El productor interno y el cliente se corresponden en forma directa con los roles de Scrum Master y Product Owner de Scrum respectivamente.

El rol de verificador beta no está presente en Scrum pero se detecta su existencia en el relevamiento de la realidad local y en la industria del videojuego en general. Su responsabilidad es la de realizar la verificación funcional del videojuego y comunicar su resultado.

2.4.2.- Ciclo de Vida

El ciclo de vida consiste en los pasos necesarios para elaborar un videojuego, y se divide en fases iterativas e incrementales que se ejecutan en forma secuencial con excepción de la fase de gestión de riesgos que se realiza durante todo el proyecto. Las cinco fases secuenciales son: concepto, planificación, elaboración, beta y cierre (véase la Figura 24). Las fases de concepto, planificación y cierre se realizan en una única iteración, mientras que elaboración y beta constan de múltiples iteraciones.

Las fases surgen como adaptación al desarrollo de videojuegos de las fases pre-game, game y post-game que presenta Scrum, donde las dos primeras coinciden con las fases de planificación y elaboración, mientras que la tercera se corresponde con las fases de beta y cierre. Esta división se realiza ya que la fase beta tiene características especiales en la industria de videojuegos. La fase de concepto no se corresponde con ninguna etapa de Scrum y se agrega ya que cubre necesidades específicas para el desarrollo de videojuegos y se idéntica su uso en la realidad local y en la industria mundial.

Los objetivos principales de cada fase son los siguientes:
Concepto: Tiene como objetivo principal definir el concepto del videojuego lo que implica definir aspectos de negocio (público objetivo, modelo de negocio), de elementos de juego (principales características, gameplay, personajes e historia entre otros) y técnicos (lenguajes y herramientas para el desarrollo). El concepto del videojuego se construye a partir de ideas y propuestas de cada rol involucrado sobre los aspectos a definir. Las propuestas se deciden a través de reuniones y se analiza su

factibilidad con pruebas de concepto. Esta fase analiza cuando se tiene el concepto validado entre todas las partes involucradas.

Planificación: Esta fase tiene como objetivo principal planificar las fases que vienen a continuación del proyecto. Para ello es necesario definir el cronograma del proyecto, junto con sus principales hitos. Luego pasaremos a conformar el equipo para la fase de elaboración de acuerdo a las necesidades técnicas del proyecto, determinando y clasificando las tareas que el equipo no pueda cumplir. Por último definiremos el presupuesto y la especificación de las características. Esto último consiste en describir, estimar y priorizar cada una de las características funcionales y no funcionales que definen el videojuego. Una característica funcional representa una funcionalidad del videojuego desde el punto de vista del usuario final, mientras que, una característica no funcional representa una propiedad o cualidad que el videojuego debe presentar. La planificación que se obtiene en esta fase es flexible, ya que en cada iteración de la fase de elaboración se puede modificar para adaptarse a los cambios y reflejar la situación actual del proyecto.

Elaboración: El objetivo de esta fase es implementar el videojuego. Para ello, se trabaja de forma iterativa e incremental, para lograr una versión ejecutable del videojuego, al ser analizado en cada iteración. Estas se dividen en tres etapas:

- Primera etapa: en esta etapa se planifican los objetivos a cumplir, las métricas a utilizar en el seguimiento, las características a implementar y las tareas necesarias para ello.
- Segunda etapa: en esta etapa se desarrollan las características planificadas a través de la ejecución de las tareas que la componen. Al mismo tiempo se realiza el seguimiento para mantener la visión y el control de la iteración en base a los objetivos planteados.
- La tercera y última implica la evaluación del estado del videojuego y de lo ocurrido en el transcurso de la iteración para actualizar el plan de proyecto respecto a la situación actual.

Con esta forma de trabajo, se puede evaluar el avance del proyecto, lo cual permite realizar cambios a tiempo y tomar decisiones para cumplir con los plazos planificados. Además, la experiencia adquirida permite mejorar la forma de trabajo en cada iteración y aumentar la productividad.

Beta: Esta fase tiene como objetivos evaluar y ajustar distintos aspectos del videojuego como por ejemplo gameplay, diversión, curva de aprendizaje y curva de dificultad, además de eliminar la mayor cantidad de errores detectados. Se trabaja en forma iterativa liberando distintas versiones del videojuego a verificar. Para ello, primero se distribuye la versión beta del videojuego a verificar y se determinan los aspectos a evaluar. Mientras esta se verifica, se envían reportes con los errores o evaluaciones realizadas. Estos reportes son analizados para ver la necesidad de realizar ajustes al videojuego. Se puede optar por liberar una nueva versión del videojuego para verificar una vez que se realizan los ajustes. El ciclo termina cuando se alcanza el criterio de finalización establecido en el plan del proyecto.

Cierre: Esta fase tiene como objetivos entregar la versión final del videojuego al cliente según las formas establecidas y evaluar el desarrollo del proyecto. Para la evaluación se estudian los problemas ocurridos, los éxitos conseguidos, las soluciones halladas, el

cumplimiento de objetivos y la certeza de las estimaciones. Con las conclusiones extraídas se registran las lecciones aprendidas y se plantean mejoras a la metodología.

Gestión de riesgos: Esta fase es realizada durante todo el proyecto, con el objetivo de reducir la aparición de problemas y minimizar su impacto. Esto se debe a que distintos riesgos pueden ocurrir en cualquiera de las fases, por lo cual, siempre debe existir un seguimiento de los mismos. Para cada uno de los riesgos que se identifican se deben establecer la probabilidad y el impacto de ocurrencia, mecanismos de monitoreo, estrategia de mitigación y plan de contingencia.

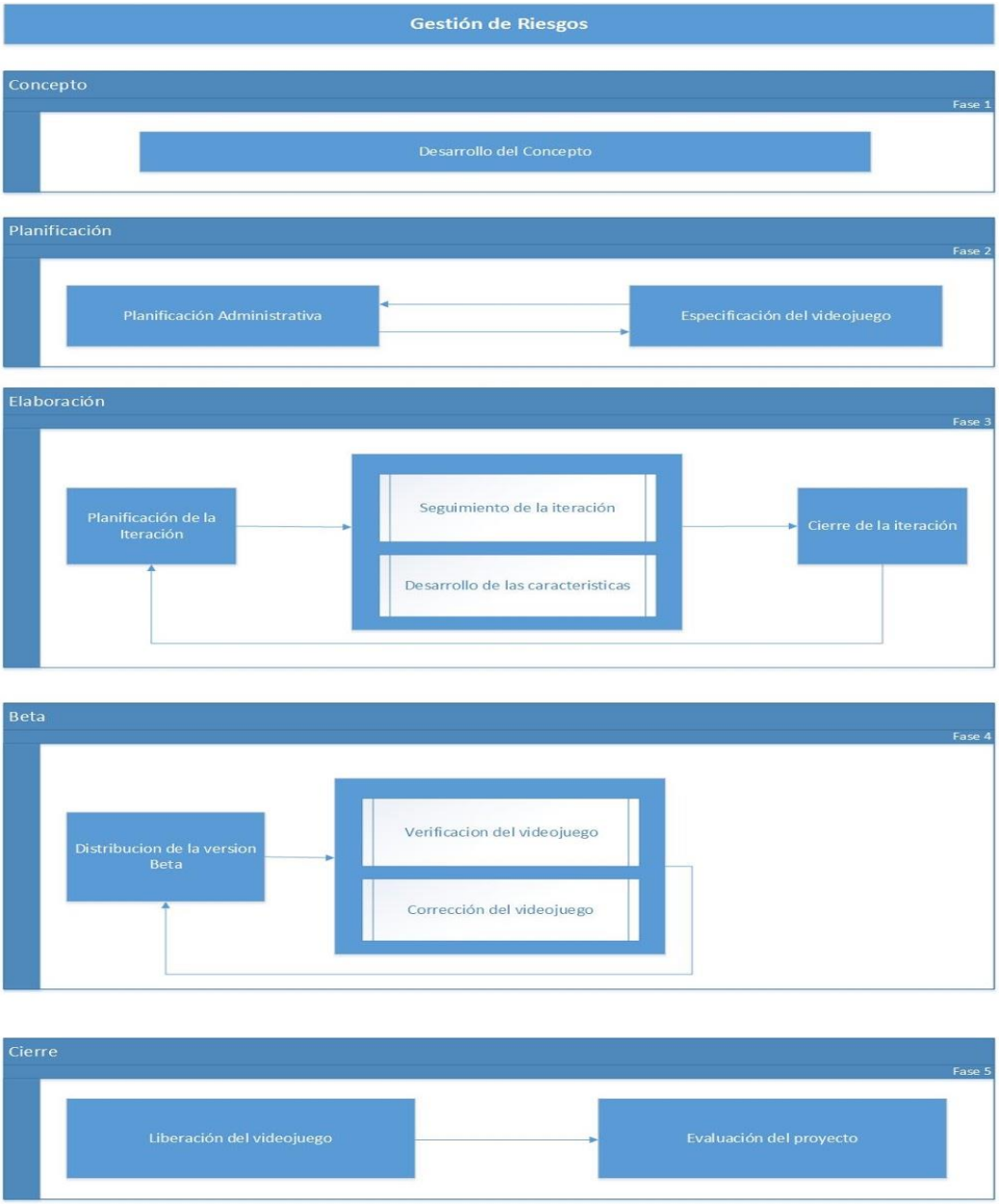


Figura 24 Fases del proceso de desarrollo.

2.4.3.- Guías

Las guías son sugerencias, pautas y herramientas para llevar a cabo en forma efectiva y eficaz las actividades que componen el proceso. A través de ellas, se incorporan a la metodología prácticas aplicadas con éxito para el desarrollo de videojuegos, además de las lecciones aprendidas con el desarrollo de cada proyecto. Actualmente SUM incluye las prácticas y herramientas de Scrum y XP, y además, artículos publicados sobre la aplicación de metodologías ágiles en el desarrollo de videojuegos.

Capítulo 3.- Objetivos y metodología

El objetivo principal del proyecto es el de realizar un estudio que englobe todo el proceso de desarrollo de un videojuego. Se ha comenzado aplicando un método de trabajo, que se ha ido siguiente hasta alcanzar nuestros objetivos. Y como base para desarrollar *IsoAbbey* se ha utilizado la herramienta *IsoUnity*.

El juego sobre el que se ha hecho la reconstrucción es *La Abadía del Crimen*. Para ello se han utilizado las herramientas de desarrollo de videojuego actuales y se ha adaptado el juego original a una versión más actualizada en diferentes aspectos. Esto no es algo trivial ya que la primera versión del juego trata del año 1987 y la concepción del videojuego en esa época es muy diferente a la actual.

Como se menciona en el primer párrafo, se ha utilizado la herramienta *IsoUnity* para el desarrollo de la reconstrucción de *La Abadía del Crimen*, siendo esta la primera vez que se utiliza con este fin. Por esta razón, será otro de los objetivos ver si esto es posible y aplicar soluciones a los problemas que vayan surgiendo. Para esta última parte contaremos con la ayuda de Víctor e Iván, desarrolladores del videojuego, para solucionar bugs de Ismael, quien ayudará en la parte de desarrollo de nuevos elementos y funcionalidades

3.1.- Motivaciones del proyecto

Las motivaciones que han llevado hacer el trabajo de fin de grado sobre una video aventura clásica han sido varias:

- La primera es la oportunidad de poder desarrollar un videojuego de estilo clásico, que es un género que no se desarrolla tanto en la actualidad.
- La segunda es el poder utilizar una herramienta de desarrollo de videojuegos, la cual ha sido desarrollada por unos compañeros de la universidad el año pasado. Además de aprender a utilizar Unity, que es una de las principales herramientas en el mercado para el desarrollo de videojuegos.
- Y la última de todas, es el desarrollar una reconstrucción basada en el mítico juego *La Abadía del Crimen*, que ha tenido un gran impacto en la época de los videojuegos de 8 bits y que actualmente cuenta con un gran número de seguidores.

3.2.- Objetivos

Se destacan los siguientes objetivos:

- Características de un videojuego actual frente a uno clásico.
- *IsoUnity* en un entorno de producción real.
- Proceso de desarrollo de un videojuego desde la concepción a su publicación.

- A continuación se irá exponiendo el plan de trabajo que se va a seguir para poder alcanzar los objetivos anteriores y se tratará de explicar porque han sido estos los objetivos elegidos.

3.3.- Plan de trabajo

Se ha concebido este TFG como un proyecto Software, por lo tanto se va a tratar de aplicar un modelo de trabajo que se pueda adaptar tanto al análisis previo que se quiere realizar, como al propio desarrollo del videojuego. Se puede considerar que el proyecto ha sido dividido en dos grandes bloques:

Primer bloque.

El primer bloque abarca el tiempo transcurrido desde Octubre hasta Junio. Engloba las tres primeras fases que define el proceso Unificado de Rational (RUP) para realizar un proyecto software. Estas fases pueden verse en la Figura 25:

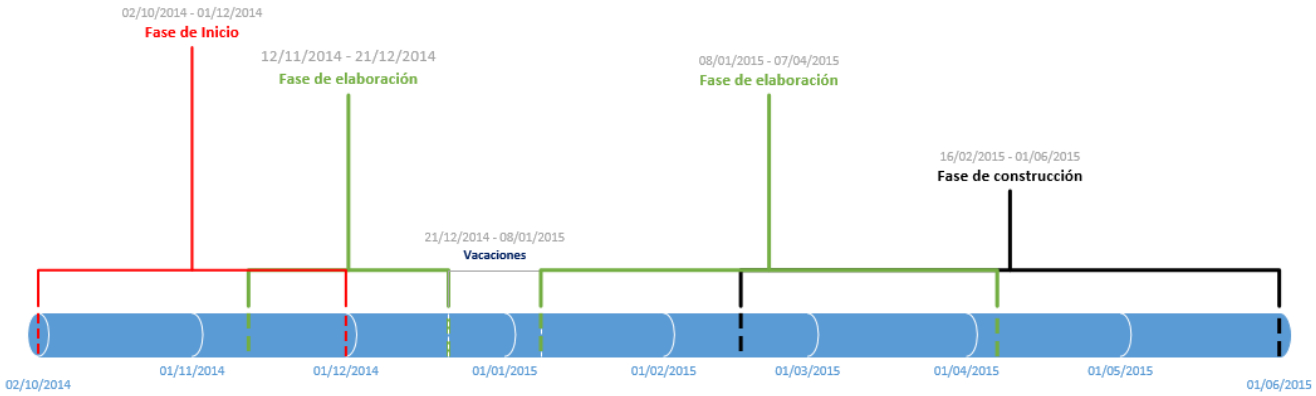


Figura 25 Escala de tiempo de las tres primeras fases RUP del proyecto

Inicio

En esta fase se ha definido el tipo de TFG que se quería realizar, tomando como referencia las especificaciones descritas en la selección de TFG de la facultad. Para ello, el tutor del proyecto Federico Peinado, ha expuesto sus ideas principales del tipo de juego que quería desarrollar, además de su alcance a largo plazo, que en este caso era la futura publicación del juego. Una vez definido qué tipo de proyecto se iba a hacer, se realizó un brainstorming para sacar varias ideas con las que empezar a trabajar.

El siguiente punto que se tuvo en cuenta en esta fase, fue la identificación de riesgos posibles. Los riesgos que se valoraron como más probables, fueron los relacionados con la herramienta *IsoUnity*, que era la que se utilizaría para desarrollar el Remake. Esto era debido a que *IsoUnity* es una herramienta en fase de testeo. A lo largo del desarrollo del videojuego han surgido los siguientes inconvenientes:

- El primer inconveniente surge al producirse continuos fallos a la hora de utilizar la herramienta *IsoUnity* y ha acompañado a toda la realización del proyecto. Esto ralentizó en gran medida el avance del mismo ya que no solo había que tratar con la propia reconstrucción, sino que también el propio motor presentaba problemas que había que solucionar. Esto tuvo su peor resultado al descartarse un mapa ya terminado por problemas con el tamaño de la escena generada (superior a 2 gigas). Finalmente los hermanos Colado trabajaron con Antonio para solucionar los problemas encontrados en *IsoUnity*.

- El inconveniente fue relacionado con la actualización del programa Unity. Unity se actualizó a la versión 5 durante el periodo de ejecución del proyecto, por lo que se decidió actualizar *IsoUnity* en paralelo. Esto provocó nuevos errores y retrasos así como la inutilización de ciertas texturas que no encajaban en la nueva versión de *IsoUnity*.

Por último, en esta fase se concretaron las reuniones. Donde se estableció que cada 15 días nos reunimos. También distinguimos entre dos tipos distintos de reunión:

- *Software.* Reuniones para solventar los problemas que iban apareciendo con *IsoUnity*, además de la forma de plantear las nuevas características que requería IsoAbbery.
- *Memoria y documentación.* Reuniones para ver cómo se va enfocando la memoria y correcciones que debían hacerse.

Para llevar un control sobre las tareas que se iban definiendo en cada reunión, se elaboró un diagrama de Gantt con la herramienta Microsoft Project (véase la Figura 26). Este diagrama ha permitido realizar un riguroso seguimiento del trabajo planificado inicialmente y detectar los momentos en los que nos desviábamos de la planificación realizada.

Las tareas que se incluyeron en el diagrama de Gantt se determinaron en las primeras reuniones. A continuación se establecería una estimación de tiempo necesario para llevarlas a cabo. En las reuniones que coincidían con la mitad de desarrollo de una tarea, se controlaba cómo iba evolucionando dicha tarea y si sufría algún retraso. En la imagen se puede observar como una tarea (color verde) tiene un grupo de objetivos (color azul) que hay que ir cumpliendo en un intervalo de tiempo.

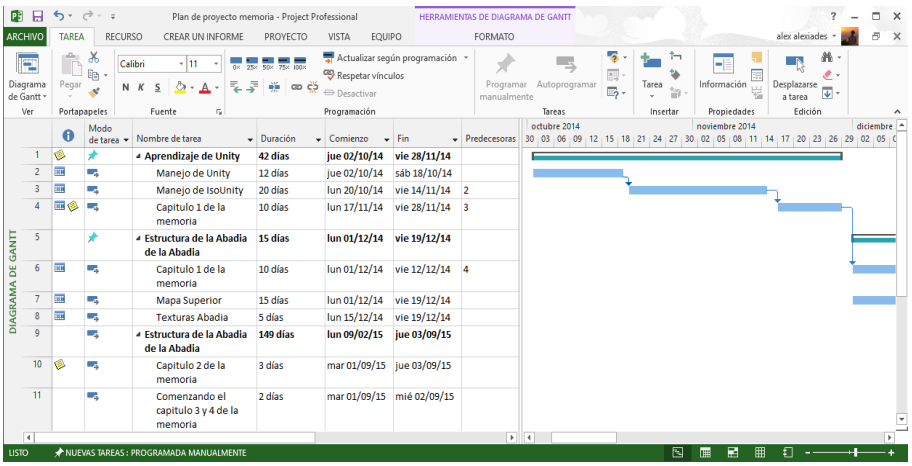


Figura 26 Diagramas de Gantt para programar las tareas del proyecto

Elaboración

En la fase de elaboración se han determinado las soluciones técnicas del proyecto. Así como en la fase de inicio, se determinaba que tipo de videojuego se iba a realizar. En esta fase se determina como se hará el videojuego. En esta fase también se elaboran los requisitos del nivel de diseño, pudiendo saber si el videojuego es técnicamente viable, así como concretar qué tipo de tecnología se va a utilizar para la fase de construcción del proyecto. Aunque el tipo de tecnología que se iba a utilizar está bastante claro en la fase inicial. Otra característica de esta fase es que no tiene punto de retorno, una vez se comienza con la fase de construcción. La falta de tiempo será un problema tan grande, que es inviable comenzar a desarrollar otro videojuego.

Esta fase se divide en dos partes:

1. Análisis, que es compuesto:
    a. Casos de uso: La elaboración de los casos de uso no iba a estar completamente definido. Ya que será un proyecto con fines educativos y de investigación sobre el funcionamiento de la herramienta *IsoUnity* a la hora de crear un videojuego. Además se sabe que irán apareciendo casos de uso de forma continua a lo largo del desarrollo del proyecto.

    b. Diagrama de Clases: Describirá las clases que se encargarán de orquestar todo y darles sentido. Este diagrama vendrá definido por las clases que usa Unity.

        Dicho diagrama permitirá saber el comportamiento de las clases de Unity e *IsoUnity*. Por ejemplo se puede deducir que la clase MapEditor deberá implementar Editor, siendo editor de la clase Map. De esta forma, todo map, al ser seleccionado, podrá ser editado en la parte del inspector, de forma similar a como se realiza con cualquier terreno de Unity.

        Para realizar el editor, las primeras pruebas que se realizaron propusieron un sistema simple, cuya funcionalidad sería almacenada en la propia clase. Sin embargo, al observarse la gran cantidad de código, variables y métodos débilmente acoplados que surgían y observarse la necesidad de que fuera fácilmente extensible, se optó por un diseño modular. Para ello, cada módulo deberá implementar MapEditorModule y el MapEditor se encargará de ponerlos en marcha cuando sea necesario. El diseño final será similar al de la Figura 27.

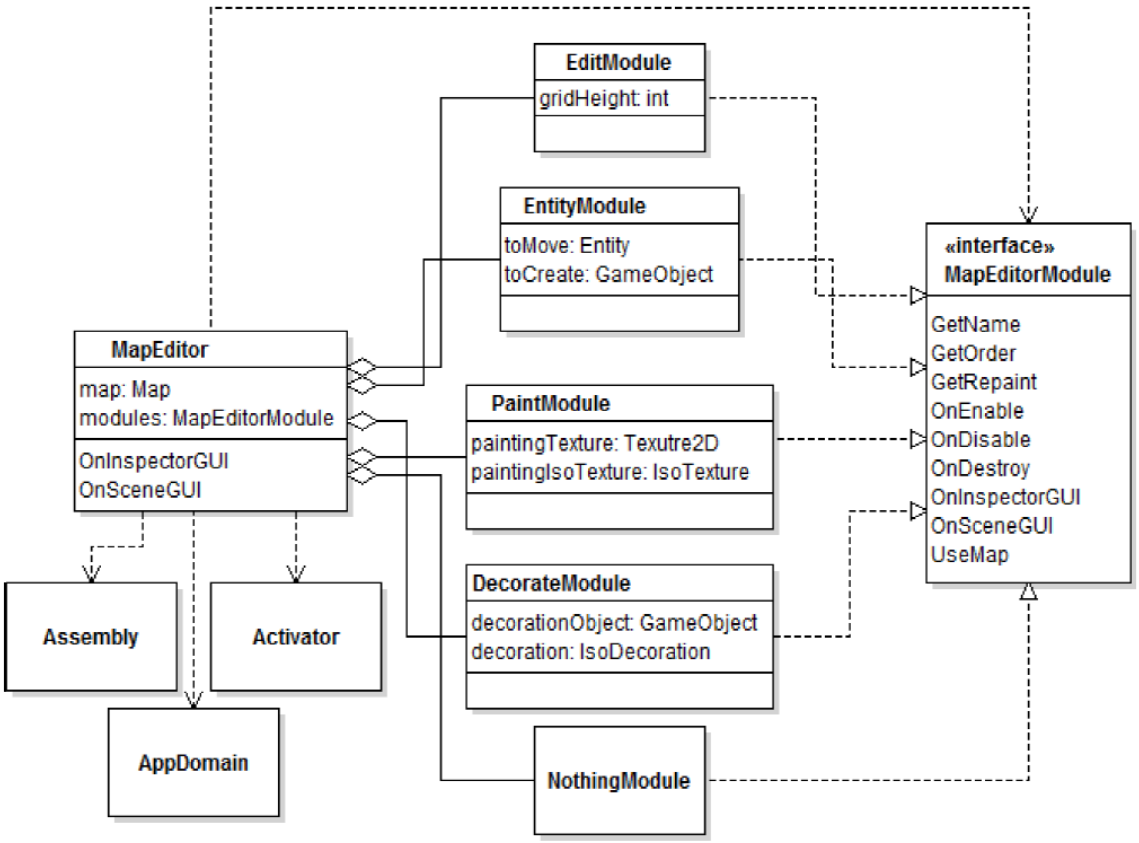


Figura 27 Diagrama de clases del editor de mapas

A simple vista, el diseño es simple y claro pero cabe matizar un hecho sobre éste. MapEditor nunca conoce sus módulos, sino que los crea dinámicamente.

El concepto de dinámicamente es un aspecto muy característico de los lenguajes interpretados (Java, JavaScript, C#...), y en este caso, se refiere a la capacidad de moverse a través de la meta información de las clases, para hallar todas las clases que implementan MapEditorModule y crear en tiempo de ejecución una instancia de éstas.

c. Diagrama de Secuencia: Nos sirve para representar el funcionamiento de forma general del lanzamiento de los distintos editores del mapa, que será el siguiente (véase Figura 3.4.):

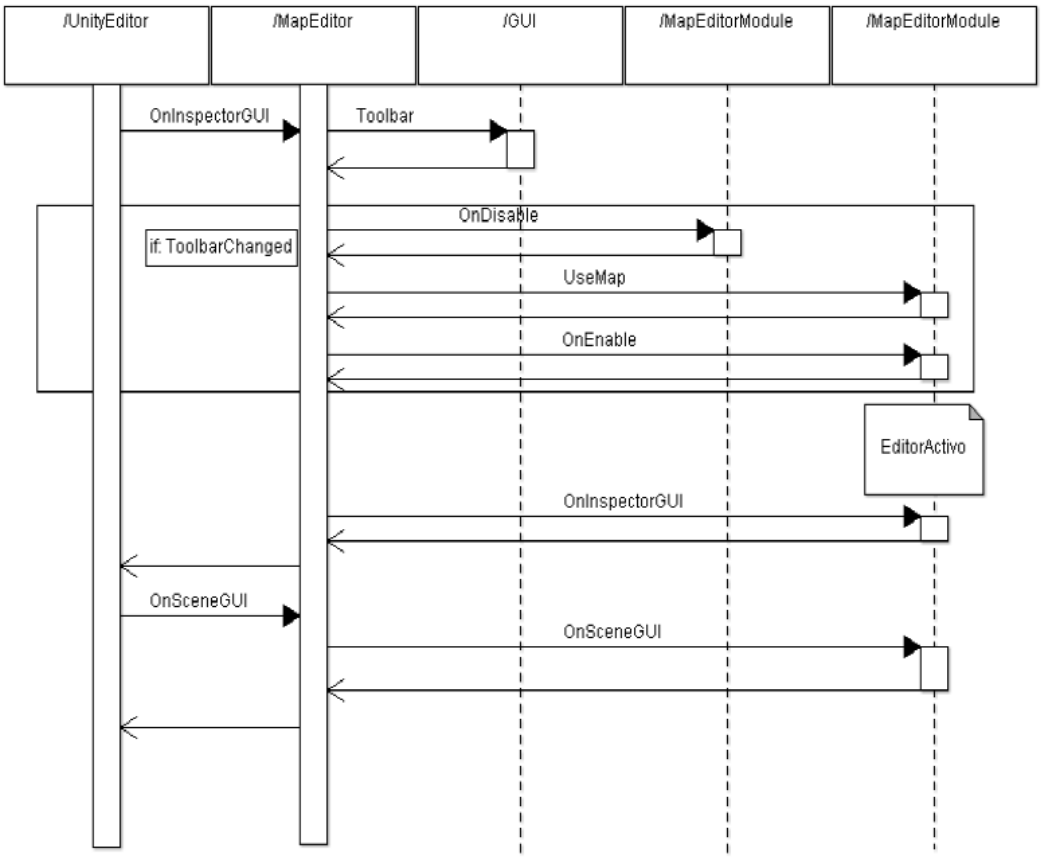


Figura 28 Diagrama de secuencia del funcionamiento general de los distintos editores del mapa

La secuencia describe el procedimiento a llevar a cabo en cada dibujado. En primer lugar, se creará una herramienta para seleccionar uno de los posibles módulos. Si se detectara el cambio de herramienta, el editor activo se desactivaría y se habilitaría el nuevo editor seleccionado, permitiendo así que cada editor pueda aprovechar este momento para adquirir datos sobre el mapa o crear objetos que ayuden como apoyo en su funcionamiento, tales como la carga en memoria de las IsoTextures o IsoDecorations.

De no detectarse ningún cambio en la herramienta, se proseguirán haciendo las llamadas sobre el anterior editor activo, sin necesidad de llamar a los métodos de habilitación y deshabilitación.

Por otro lado, la implementación de cada editor se verá en profundidad en el próximo capítulo, ya que debido a la gran vinculación con la plataforma, es difícil conocer en este punto las dependencias exactas con las clases de Unity de cada uno de ellos. Las funcionalidades que deberán realizar los módulos, sin embargo, fueron descritas en el subcapítulo anterior, por lo que las definiciones y mecánicas a implementar se tomarán como referencia para llevar a cabo la implementación.

2. Diseño, que se compone de los siguientes elementos:

a. Desarrollo de la Arquitectura Software: *IsoAbbey* tiene que basarse en la arquitectura base de *IsoUnity* y a partir de aquí crear los elementos propios de la reconstrucción manteniendo cierta coherencia con las clases ya existentes. Para poder extraer los detalles del juego original se decidió que la mejor forma era mediante la experiencia propia, por lo que los miembros del grupo obtuvieron sus propias conclusiones y posteriormente pusieron estas en común. Tras esto se llegó a varias elementos que habría que construir:

i. Los controles: Como manejar a los personajes.

ii. El sistema de cámara: Referente al establecimiento de una cámara isométrica fija en lugar una que cambiase de dirección propia del original.

iii. La interacción con los personajes del juego: En la versión original la interacción con NPCs es casi inexistente, esta característica tiene que cambiarse en pos de actualizar el juego a una versión interesante en la actualidad.

iv. El comportamiento de los personajes en el juego: Hay que tratar de reconstruir el comportamiento de cada uno de los personajes, pues son todos diferentes.

v. Sistema de horarios: El sistema de juego está dividido en 7 días, estos se dividen a su vez en las horas canónicas (horas que representan lo que un monje debería hacer en un monasterio, son franjas horarias). Ciertas características de la abadía deben cambiar en función de la hora en la que el juego se encuentre. Se encuentra una explicación más detallada de esto en el Anexo 2.

b. Desarrollo de la Interfaz del Juego: El Interfaz de *IsoAbbey* está basado en el HUD del juego original, se tomó esta decisión para poder mantener cierta cohesión entre ambas versiones y despertar la nostalgia de los jugadores. Ciertos elementos del interfaz original fueron modificados para hacerlos más atractivos al jugador, esto se hizo tomando ideas de otros videojuegos. El cambio más sustancial añadido se basa en la posibilidad de selección del personaje a controlar.

Construcción

La fase de construcción ha sido la más larga de todas, ya que al trabajar con una herramienta en fase de pruebas, se tuvo que tratar con una enorme cantidad fallos, los cuales produjeron ralentizaciones a la hora de ir desarrollando *IsoAbbey*. Para solventar semejante cantidad de errores se ha contado con el soporte de los hermanos Pérez Colado y la inestimable ayuda de Ismael. Este soporte ha tenido lugar a través de la herramienta GitHub. A través de un repositorio de *IsoUnity* se han podido ir reportando las incidencias con su posible solución o lugar en el que ocurren para poder ser arregladas y con ello actualizar *IsoUnity* a una versión más estable.

Las fases de construcción y elaboración se han desarrollado en paralelo, debido a que se han ido definiendo nuevas características del juego a medida que se iba avanzando con el desarrollo. Además, esta fase también se ha caracterizado por tener dos tipos de desarrollos:

1. Implementación del código actual en C#. Implementar algunas clases de *IsoUnity* para poderlas adaptar la jugabilidad y características de *IsoAbbey*.
2. El montaje del juego con la herramienta IsoUnity. Este tipo de desarrollo era el que iba a predominar al principio del proyecto (véase la Figura 29). Debido a los

problemas con la herramienta que han ido apareciendo, el esfuerzo destinado a montaje se ha reducido drásticamente en pos de la implementación.

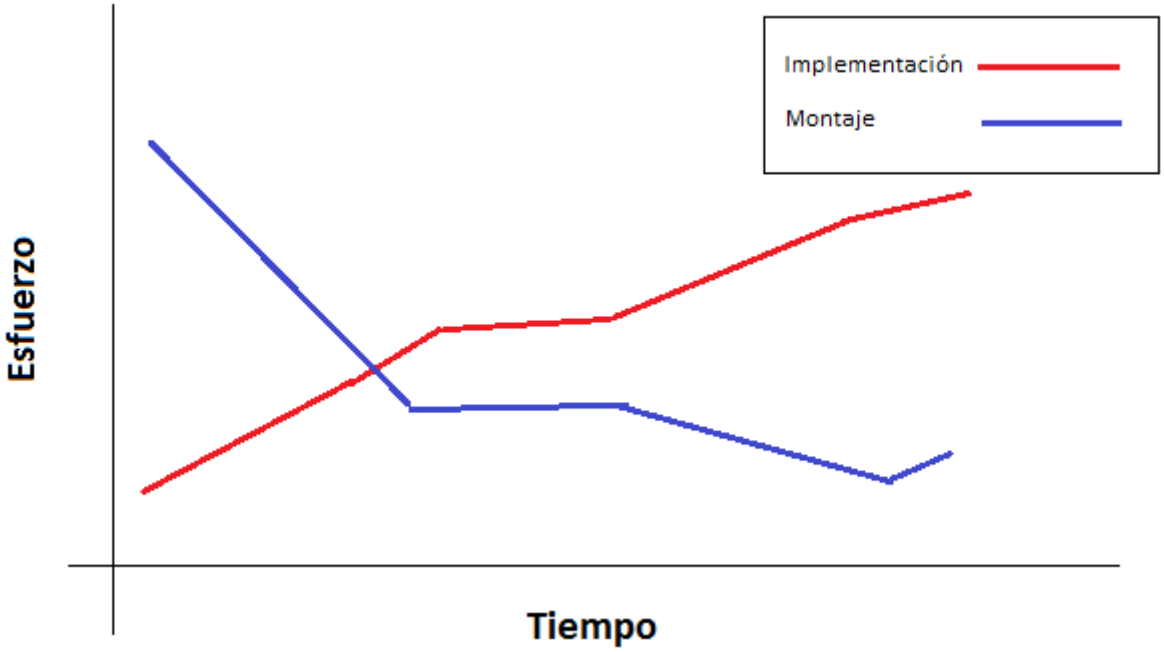


Figura 29 Gráfico de esfuerzo a lo largo de la fase de elaboración

Segundo bloque.

El segundo bloque comprende desde Junio hasta la fecha de entrega del proyecto en Septiembre. Al ser pocos meses y a falta de terminar parte de la fase de construcción del proyecto y completar la memoria correspondiente a dicha parte se ha empleado una metodología ágil. Dicha metodología es similar a SCRUM, ya que se ha ido haciendo el proyecto de forma incremental y con revisiones de forma iterativa y el grupo está formado únicamente por dos estudiantes. (Véase la Figura 30).

Figura 30 Metodología empleada en el proyecto

La organización de las reuniones ha tenido lugar de una forma distinta a las del primer bloque, debido a que al ser verano la facultad no estaba disponible como lugar de reunión. Por ello se optó por realizar videoconferencias con la aplicación hangout de google. El método de organización de los objetivos semanales, también ha sido cambiado en pos de una nueva herramienta llamada Waffle (véase la Figura 31). Esta herramienta ha permitido gestionar y hacer un seguimiento de los objetivos que se tenían que cumplir de una forma intuitiva, rápida y fácil.

Figura 31 Herramienta Waffle para el control de objetivos

3.4.- Progresos previstos para *IsoUnity*

La maduración de *IsoUnity*, como la de cualquier herramienta software compleja, es un proceso extenso debido a lo sofisticado de su estructura y funcionalidad, ya que está desarrollado como parte de la herramienta Unity, muy integrada dentro de ella.

Uno de los objetivos que también nos planteamos en este trabajo es que *IsoUnity* continúe progresando y que nuestro trabajo fomente la realización de mejoras en la herramienta, aumentando su estabilidad y permitiendo crear juegos de una manera más fluida, con menos errores y de manera más intuitiva y natural para el desarrollador.

Esta carencia de estabilidad que presenta actualmente la herramienta se ha ido notando a lo largo de la elaboración del proyecto. Otra dificultad que se ha encontrado es la falta de características disponibles o la limitación de algunas de las ya existentes. Esto ha llevado a la creación de elementos nuevos y módulos paralelos en espera a que se mejorasen los "oficiales" que ya existían en *IsoUnity*.

Capítulo 4.- *IsoAbbey*: Análisis y diseño de una video-aventura clásica

El objetivo principal de este capítulo es detallar cómo se ha desarrollado *IsoAbbey* a partir de un análisis previo de *La Abadía del Crimen*. Todo esto comenzó con la recogida de documentación sobre el juego original. Más adelante ha sido necesario jugar a dicho videojuego para tomar consciencia de a que enfrentarse. Por último, se ha hecho un análisis del juego, para ver qué aspectos se deben incluir en la reconstrucción y cuales tienen que cambiarse para adaptarse al modelo de videojuego actual.

4.1.- Análisis de La Abadía del Crimen

A la hora de analizar el juego de *La Abadía del Crimen*, se han ido revisando diferentes aspectos del juego. A continuación se establece una definición sobre los aspectos más relevantes del juego:

En primer lugar debemos decir que se trata de un juego de un solo jugador. El jugador se reencarna en fray Guillermo de Occam y en el joven novicio Adso de Melk, quienes serán los encargados de investigar una serie de asesinatos que suceden en una abadía benedictina, situada en la península de Italia.

Se caracteriza por no ser un juego sencillo, debido a que no se puede guardar la partida y que a la hora de realizar los objetivos hay que ser extremadamente precisos, puesto que el juego penaliza el mínimo error.

El juego se desarrolla a lo largo de un espacio de tiempo de 7 días, los cuales están subdivididos cada uno en franjas horarias, las horas canónicas, es decir, éstas son la organización real que debe seguir un monje en el claustro.

Una vez vistos los aspectos más característicos del juego se va a proceder a analizar las distintas partes que componen el juego de *La Abadía del Crimen*.

- Desarrollo de la trama del juego
- Acciones generales de los personajes
- Jugabilidad
- Cámara del juego, diseño y ambientación
- Diseño de los personajes de la abadía
- Extracción de texturas de la abadía

4.1.1.- Desarrollo de la trama del juego

El desarrollo del juego, como se comenta en el apartado anterior, transcurre a lo largo de 7 días. Aunque los días inicial y final no son enteros. Cada día a su vez se divide en 7 franjas horarias que coinciden con las horas canónicas de un monasterio. Estas horas son: Noche, prima, tercia, sexta, nona, vísperas y completas. La hora en la que se esté jugando influirá en el entorno de la abadía, es decir, los monjes actuarán de forma diferente y el escenario también sufrirá ciertos cambios. En el anexo 1 se ha añadido una descripción detallada de lo que va sucediendo en cada día. Este análisis es realizado para cada una de las horas, además se adjunta una pequeña descripción de lo que sucede en ese momento.

4.1.2.- Acciones generales de los personajes

Abad

	**Monje superior de la abadía**

- Si no se obedecen sus órdenes, nos sanciona.
- Si el primer día Guillermo visita el ala izquierda de la abadía, el abad lo echa.
- Si Guillermo visita la biblioteca cuando no es de noche o prima, el abad lo va a buscar a la entrada de la biblioteca y lo echa.
- Si el abad descubre a Guillermo en su celda, lo expulsa.
- Va a buscar a Guillermo si entra en su celda.
- Si es advertido por Berengario, va a pedir el pergamino a Guillermo.
- Tras quitar el pergamino a Guillermo, va a dejarlo al escritorio de su celda.
- Si Malaquías o Berengario van a advertirle, va a su celda a esperarles.
- Si Bernardo va a darle el pergamino, va a la entrada de la iglesia a esperarle.

Guillermo

	**Fraile franciscano protagonista del juego**

Controles:

ARRIBA: Anda hacia adelante
DERECHA: Gira a la derecha
IZQUIERDA: Gira a la izquierda
ABAJO: Controla a Adso, que andará en la dirección que mira Guillermo
ESPACIO: Deja un objeto

Adso

	**Novicio ayudante de Guillermo**

A no ser que tenga que asistir a los oficios, ir al refectorio o a su celda a dormir, sigue a Guillermo y puede ser controlado pulsando ABAJO; andará en la dirección en que mira Guillermo.

Frases que dice cuando está cerca de Guillermo:

- Queda poco para que amanezca: *"Pronto amanecerá"*
- Queda poco aceite: *"La lámpara se agota"*
- Se agota el aceite: *"Se ha agotado la lámpara"*
- Andan demasiado tiempo sin luz: *"Jamás conseguiremos salir de aquí"*
- Entran en la biblioteca sin lámpara: *"Debemos encontrar una lámpara, maestro"*

Berengario

	**Uno de los monjes que trabaja en el Scriptorium**

- Si Guillermo coge el pergamino en su presencia, le ordena que lo deje o advertirá al abad.

"Dejad el manuscrito o advertiré al abad"

- Si Malaquías le ha dicho a Guillermo que Berengario le puede mostrar el scriptorium, busca a Guillermo y le dice:

"Aquí trabajan los mejores copistas de occidente"

Luego se dirige a la mesa de Venancio y dice:

"Aquí trabajaba Venancio"

Bernardo

	**Religioso dominico representante del tribunal de la Santa Inquisición local**

- Si no tiene nada que hacer, se va a dar paseos por la abadía.
- Si Guillermo tiene el pergamino y pasa cerca de él, se lo pide diciendo:
  "Dadme el manuscrito, Fray Guillermo"
- Si Guillermo ha dejado el manuscrito en algún lugar, va a buscarlo.
- Si consigue el pergamino, busca al abad para dárselo.

Malaquias

	**Monje encargado de la biblioteca**

- Se queda quieto si el abad busca a Guillermo para echarle de la abadía.

- Si Guillermo se acerca a la entrada de la biblioteca en su presencia, bloquea la entrada y dice:

"Lo siento, venerable hermano, no podéis subir a la biblioteca"

Severino

	**Monje herbolario**

- Si no tiene nada que hacer se da paseos desde su celda hasta el lugar donde nos presentan a Jorge y viceversa.

- Intentará presentarse a partir del segundo día diciendo:

"Venerable hermano, soy Severino, el encargado del hospital. Quiero advertiros que en esta abadía suceden cosas muy extrañas. Alguien no quiere que los monjes decidan por si solos lo que deben saber"

(Al terminar, se irá a su celda)

4.1.3.- Jugabilidad

En esta se hace un análisis sobre la jugabilidad del juego y de cómo se ha adaptado o mejorado dicho jugabilidad al prototipo desarrollado.

Menú principal

En el juego original

La Abadía del Crimen nada más comenzar muestra la carátula del juego y no permite que se elija ningún tipo de opción.

En IsoAbbey

En la carátula principal del juego se ha añadido un menú con varias opciones. Este menú es algo necesario, ya que da al jugador cierto control sobre el juego y la información necesaria para poder jugar, por ejemplo un pequeño tutorial.

Movimiento simultáneo de personajes.

En el juego original

El sistema para manejar los dos personajes al mismo tiempo es muy complejo, ya que los controles son algo confusos. Adso se podía mover pulsando una tecla, y el personaje caminaba en la misma dirección a la que mirase su maestro, salvo que hubiera algún obstáculo que tratara de esquivar o si llegaba al final de la pantalla, que retrocedía un poco (aunque se le puede insistir para que siguiese avanzando en dicha dirección). Mientras se movía a Adso se podía seguir moviendo a Guillermo con normalidad, pero con la orientación de Guillermo haciendo que Adso no pueda ir a otra sección de la abadía.

En *IsoAbbey*

El sistema de selección de personajes que se ha implementado se puede observar en la Figura 32. Hay que seleccionar la Figura del personaje que se desea controlar (Guillermo o Adso) y se puede controlar a cada uno de forma independiente. Controlando a Guillermo, Adso se limita a seguirnos y permanecer cerca (aunque esto se puede desactivar en función de las necesidades de la aventura). El modo de control de Adso sólo funciona cuando no hay que ir ni a comer ni a misa, y permite moverle libremente, dejando a Guillermo quieto donde está (también se puede hacer que Guillermo siga a Adso). Si se pierde el control de Adso éste siempre tratará de volver con Guillermo.

También se permite la coexistencia de dos inventarios. Ambos pueden recoger objetos, que se muestran en la misma barra, pero solo pueden ser utilizados por aquel que los recogió. Si la ocasión lo requiere, también se puede soltar un objeto con un personaje y recogerlo con el otro.

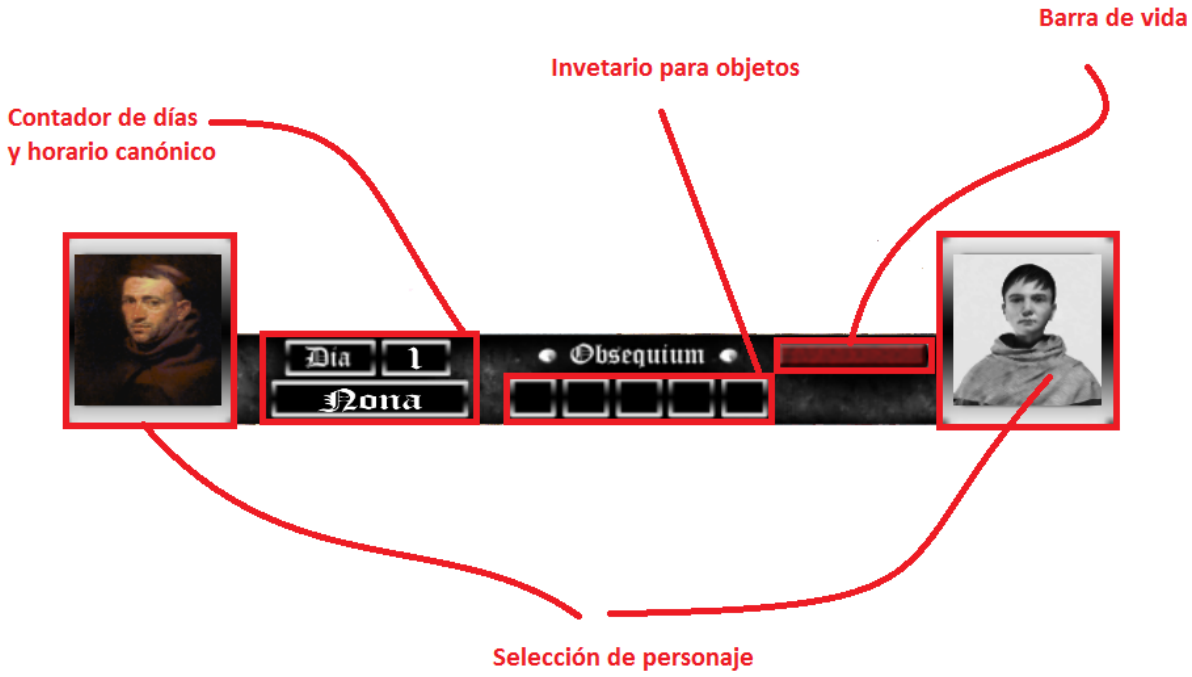


Figura 32 Panel de control del juego en el pie de página del juego

Movimiento del personaje

En el juego original

Guillermo solo permitía un movimiento hacia delante o hacia atrás, se podía seleccionar una nueva dirección estáticamente y después moverse en dicha dirección. Esto era así para poder mantener la dirección del movimiento ya que la cámara se cambiaba automáticamente al cambiar de estancia.

En *IsoAbbey*

Funciona a través de las teclas WSAD, pulsando en algún lugar de la pantalla con el ratón o a través de un mando. Mover los personajes usando el ratón es útil para trasladar el juego a dispositivos táctiles (tocar o clicar sobre dónde se desee que vaya el personaje). Esto puede hacerse ya que *IsoAbbey* dispone de una cámara fija.

Persecución del abad

En el juego original

Un problema que tiene el juego es a la hora de tener que seguir a algún, esto se puede verse en el abad al principio del juego, ya que este se dedica hacer movimientos aleatorios y a llamar la atención del jugador en cuanto éste deja de seguirle, restándole con ello puntos de vida (“Obsequium”).

En *IsoAbbey*

Igualmente se debe seguir al abad con Guillermo. Pero el recorrido del abad tiene una forma más lógica, se han aumentado los tiempos de espera del abad, los cuales se cancelan automáticamente en cuanto el jugador se encuentra a una distancia determinada. Todo esto ha hecho posible que el juego sea más amable con el jugador.

Diálogos en el Juego

En el juego original

No hay diálogo. Simplemente monólogos que se tienen que leer independientemente del lugar en el que el jugador se encuentre, ya que salen en un display dentro del HUD. Existen únicamente un par de momentos puntuales que pueden considerarse como diálogo, un ejemplo es cuando se hace de noche y Adso pregunta al jugador si este desea ir a dormir, a lo que se puede contestar con un "Si o No".

En *IsoAbbey*

Hay un sistema de diálogo que funciona en tiempo real, el tiempo que tarda en ocurrir en diálogo resta tiempo de lo que queda de la Hora Canónica actual. Existen dos tipos de diálogo: Por un lado está el monólogo mientras se anda (Véase la Figura 33), en los que otros personajes (o tus propios personajes) hablan pero no interrumpen la acción. Se suelen mostrar típicamente diálogos pero en ocasiones se da la opción de seleccionar varias respuestas, aunque no tengan repercusión real sobre el juego, para que el personaje que controlas pueda responder (usando ratón o el dedo sería simplemente pinchar sobre la opción; con teclado habría que tener una tecla para cambiar de opciones y otra para seleccionar esa sí puede ser la misma que usar objeto, el enter, por ejemplo). Por otra parte se fuerzan los diálogos en los que se inhabilita el movimiento del personaje (se permite que se mueva el otro personaje, en el caso de que la acción no vaya con él para resolver ciertos puzles), que serían los más habituales (en ese caso las propias flechas arriba y abajo podría permitir elegir opción, que es lo más normal en una aventura gráfica). En ambos casos, se consideran personajes distintos que participan en el diálogo a Guillermo y a Adso.

Figura 33 Mensajes sobre una estética de pergamino en el juego

Estructura de tiempos.

En el juego original:

El "timing" o las pruebas cronometradas son muy importantes, siendo fundamental llegar a comer o a misa en tiempo, y colocarse en un lugar exacto o se irán recibiendo amonestaciones que irán restando Obsequim. Y esta es la única forma de perder puntos de vida en el juego, no se incluye ningún tipo de misión secundaria que permita la ganancia de puntos de vida ni hay ningún tipo de penalización extra.

En *IsoAbbey*

El poder manejar de forma no simultáneamente, pero de manera concurrente a Guillermo y Adso, puede ampliar enormemente las opciones de juego. Por ejemplo con la construcción de puzles que sólo se pueden resolver haciendo unas acciones con un personaje y otras con otro de manera coordinada. Un ejemplo de esto que se ha desarrollado es: Abrir una puerta que requiere que un personaje esté apoyado sobre una baldosa y el otro sobre otra. O dejar a un personaje bloqueado haciendo fuerza sobre una palanca (por ejemplo Guillermo, que sólo puede aguantar su peso 10 segundos) y mientras tanto Adso tiene que recorrer un pasillo -que le lleva 8 o 9 segundos recorrer- para poner una calavera justo en una ranura que está abierta esos 10 segundos, y así atascar esa ranura para luego poder pasar por allí. Se considera que este nuevo sistema

puede aumentar las opciones del juego y parecerse a otros títulos tales como *Head over Heels.*

Interacción del juego con personajes.

En el juego original

No se encuentra diálogo en el juego original.

En nuestra versión

Se han diseñado ciertos puzles que consistan en decir primero algo con Adso, es decir, en un determinado momento de la conversación podríamos tener la posibilidad de decir A, B y C con Guillermo o pasar el control a Adso y tener la posibilidad de decir D, E y F. Sean frases parecidas (la mayor parte del tiempo, y equivalente en cuanto al flujo de la conversación) o no (ampliando las opciones del diálogo y concurriendo en nuevos flujos de la trama), la posibilidad de controlar a dos personajes permite una flexibilidad mayor a la hora de hacer que el juego de la abadía sea menos lineal.

A modo de ejemplo vamos a describir una situación en el juego: el viejo Adelmo puede decirle algo comprometedor al muchacho: "Eres tan bello como una muchacha" y aunque las opciones por defecto de Adso no son muy buenas "Cállese, viejo verde!", si cambias a Guillermo, aparecen opciones bastante mejores para salir del paso sin ofender como "Gracias, Adelmo, aunque no es la belleza del muchacho lo que me han confiado, sino su formación intelectual".

Interacción con objetos en el juego.

En el juego original

Para robarle las llaves del pasadizo a Malaquías, debido a que las tiene en su mesa y las vigila, lo que hay que hacer es controlar a Adso, ya que si Guillermo se le acerca no le deja, pero a Adso sí, ya que se supone que es más escurridizo.

En *IsoAbbey*

Hay que acercarse con Guillermo para hablar con Malaquías y le pide que le acompañe para ver un pergamino. En ese momento se puede ver una conversación con él, que normalmente es muy escueta y al terminar él se vuelve automáticamente a su "puesto de guardia". Pero sin embargo conversando varias veces se llega a descubrir que cuando se habla de su autor favorito, Platón, el comienza un monólogo. Es en ese momento cuando se puede seleccionar a Adso para poder coger sus llaves.

Función de Adso

En el juego original

Adso siempre seguirá a Guillermo, salvo en los momentos que toque ir a comer o a misa, en los que toma el papel de guía hacia el evento.

En *IsoAbbey*

Se puede tener a Adso "bloqueado" con las visiones de las gárgolas en la Iglesia, o con la escena de la muchacha (haciendo el amor en la cocina con ella). En estas ocasiones se puede manejar a Guillermo, que debe aprovechar para hacer ciertas cosas…

Modo exploración

En el juego original

Inexistente en el juego original

En *IsoAbbey*

Esta es una de las características que más se ha echado en falta en este juego. Esto es debido a que en la actualidad premia en el mercado los juegos de libertad total, para poder investigar a fondo los escenarios. El juego original de *La Abadía del Crimen* no lo permitía por lo que se ha incluido en esta versión. Este modo, al que se accede desde el menú de inicio, además de permitir al jugador investigar con libertad, incluye los ciclos de horas propios de un convento, lo que permite al jugador la inmersión dentro de la abadía.

Sonido

En el juego original.

La banda sonora de *La Abadía del Crimen* está compuesta por una única melodía que se repite durante todo el juego.

En *IsoAbbey*

Existe un módulo de sonidos donde que van cambiando la banda sonora del juego. Esto es importante ya que ayuda a meter al jugador en situación cuando alguno de los personajes se encuentra ante una situación de peligro o reproduce sonidos ante hechos importantes.

Resolución de Pantalla

En el juego original.

La resolución de pantalla, en comparación con la actual era bastante peor. Eso era debido a que la resolución dependía de la plataforma a la que estuviera adaptado el juego. Por ejemplo la resolución de los juegos para Amstrad era inferior a Spectrum.

En *IsoAbbey*

Se dispone de distintas resoluciones. Al igual que el juego original es recomendable adaptar la resolución según el dispositivo ya que no es lo mismo la resolución de un móvil que la de un PC.

Plataformas del juego

En el juego original.

La *Abadía del Crimen* se diseñó únicamente para los sistemas de ordenador doméstico más relevantes de la época: Spectrum, MSX y PC.

En *IsoAbbey*

Se puede exportar este videojuego a todas las plataformas permitidas por Unity y esto se hace de forma automática, lo que favorece al desarrollador en contra de lo que ocurría antiguamente, que había que cambiar de versión dependiendo de la plataforma a la que iba dirigido.

4.2.4. Cámara del juego, diseño y ambientación.

Para el diseño de la abadía, fue preciso que Menéndez desarrollara un programa para que Juan pudiera crear las imágenes del videojuego en el computador, ya que en aquella época no se disponía de las herramientas de diseño y modelado de imágenes que hay hoy. Se podría decir que también se ha utilizado una herramienta no comercial para tratar con el diseño de la abadía. Pero al ser una reconstrucción que guardara parte de la esencia del juego original no se ha modificado el diseño de las texturas, decorados y personajes, pero sí ha hecho falta adaptarlos a la herramienta *IsoUnity*.

Para poder realizar el diseño de *La Abadía del Crimen*, Juan se inspiró en cuatro abadías de diferentes países; Francia, Alemania, España e Italia. Los planos al igual que los personajes fueron dibujados en hojas cuadriculadas. El diseño del plano de la abadía se compone de tres escenarios, la planta principal, el Scriptorium y la biblioteca. Pero el plano final fue más grande y por falta de memoria se tuvieron que quitar varias partes (véase la Figura 34).

Figura 34 Plano original de La Abadía del Crimen en el que se basa el mapa desarrollado para nuestro Remake

El inicio del juego también ha sido actualizado. En el juego original se muestra un pergamino que se va rellenando con el prólogo del libro "El Nombre de la Rosa". Mientras que en esta nueva versión se ha utilizado un pergamino de fondo sobre el que se va insertando texto y se muestran botones que permiten: Mostrar el siguiente fragmento o saltar el prólogo. Esto puede verse en la Figura 35.

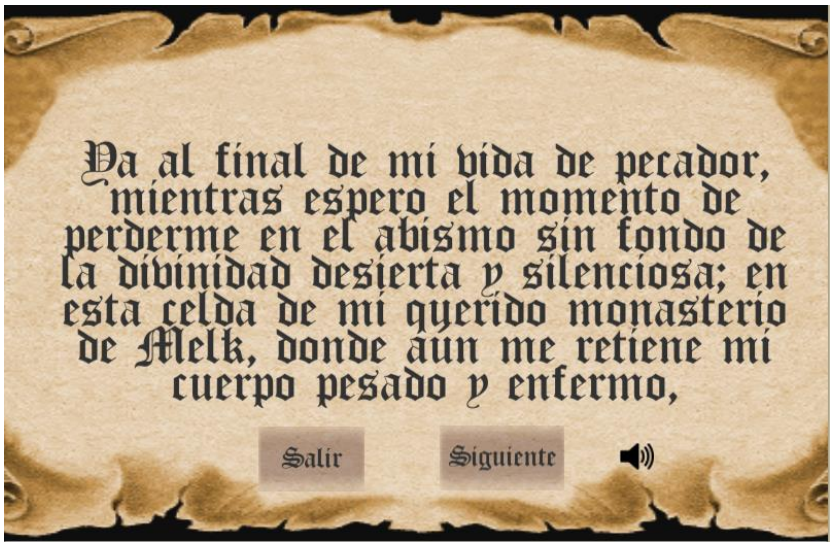


Figura 35 Introducción inicial al lanzar la partida (Prólogo)

Una cosa por la que destacó la abadía, fue por lo novedoso de la perspectiva de la cámara Isométrica. Juan utilizó este tipo de vista, que es empleada en arquitectura, para dar al jugador una sensación de profundidad. En *IsoAbbey* se ha utilizado la cámara ortogonal que proporciona Unity para darle a los juegos 3D un efecto isométrico. En la Figura 36 se puede apreciar la sensación de perspectiva isométrica que da la cámara de Unity.

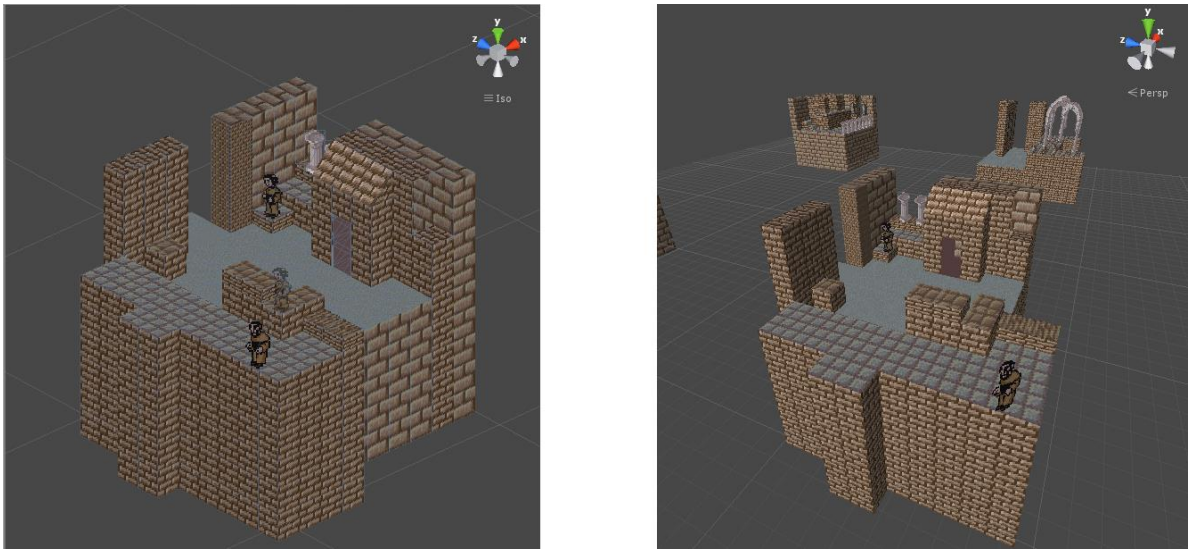


Figura 36 Cámara ortogonal en la imagen de la izquierda y perspectiva 3D libre en la de la derecha

Otra característica que cabe destacar es la posición de la cámara en cada habitación. Esto producía un efecto en el juego que era desconocido hasta entonces. Daba una

sensación al jugador de estar andando por dentro de la abadía, ya que experimentaba planos diferentes que combinaba con las habitaciones contiguas. *IsoAbbey* está construido de forma diferente, ya no está compuesto por habitaciones de 8x8, sino que ahora trata de mantener unas proporciones realistas en un mapa continuado. Esto puede verse en la Figura 37

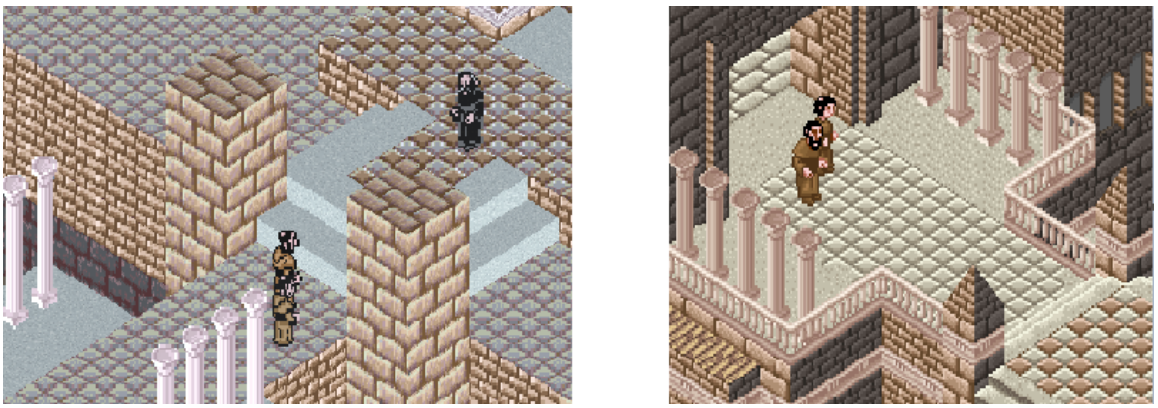


Figura 37 Estancia de nuestro Remake y misma imagen en el juego original a la derecha

El último aspecto a resaltar del sistema de cámaras, es el seguimiento al personaje principal. En el juego original, al igual que en *IsoAbbey*, la cámara sigue a Guillermo mientras éste es controlado, pero si se selecciona a Adso la cámara no cambia. En *IsoAbbey* la cámara deja de seguir a Guillermo en favor Adso. No siempre en el juego original la cámara sigue a Guillermo, sino que hay ciertas escenas en la que se enfoca a otro personaje. A Continuación se menciona a que personajes y en qué momento.

Berengario:
- Va a advertir al abad
- va encapuchado a por el libro y está entrando al *Scriptorium*
- cuando muere

Abad:
- Va al refectorio
- va a dejar el pergamino a su celda
- va a pedirle el pergamino a Guillermo
- va a expulsar a Guillermo de la abadía

Malaquías:
- Va a buscar al abad
- en Vísperas va a cerrar las puertas, o va a la cocina

Severino:
- Va en busca de Guillermo

Bernardo:
- Va a darle el manuscrito al abad

En el caso de *IsoAbbey*, debido a que está implemento un único día, no se recogen todas estas, solo aparece una de Malaquías.

4.1.5.- Personajes de *La Abadía del Crimen*

Los sprites de los personajes del juego fueron diseñados con una hoja cuadriculada. Ya que al ser gráficos de 8 bits se podía bocetar bien los personajes en este tipo de papel (véase la Figura 38):

Figura 38 Diseño original de Guillermo y Adso

Más adelante, para otros remakes se utilizaron otros tipos de programas más avanzados como el Deluxe Paint para retocar los personajes. En el caso de *IsoAbbey* se ha utilizado el Photoshop CS6 para crearlos. Las plantillas han sido generadas a partir de un sprite de Guillermo encontrado en internet, esto puede verse en la Figura 39.

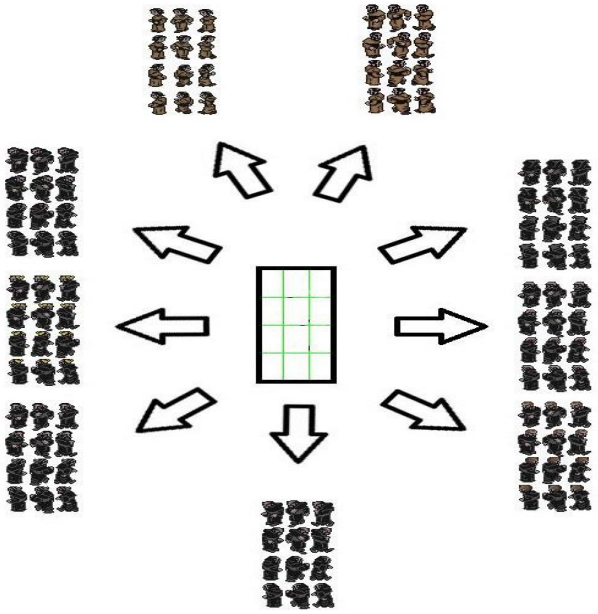


Figura 39 Plantilla de personajes que utiliza IsoUnity para crear los sprites

4.1.6.- Extracción de texturas de *La Abadía del Crimen*

En este apartado se hablará de cómo se han extraído las texturas del juego original. Para ello se ha utilizado la herramienta TextureAssistant (véase la Figura 40) que es un componente que pertenece a *IsoUnity*. Se puede localizar dentro del programa Unity en la ruta Windows > TextureAssistant. Pero antes se tienen que las imágenes de algún sitio, en el caso de *IsoAbbey* se tomaron las texturas del mapa del.

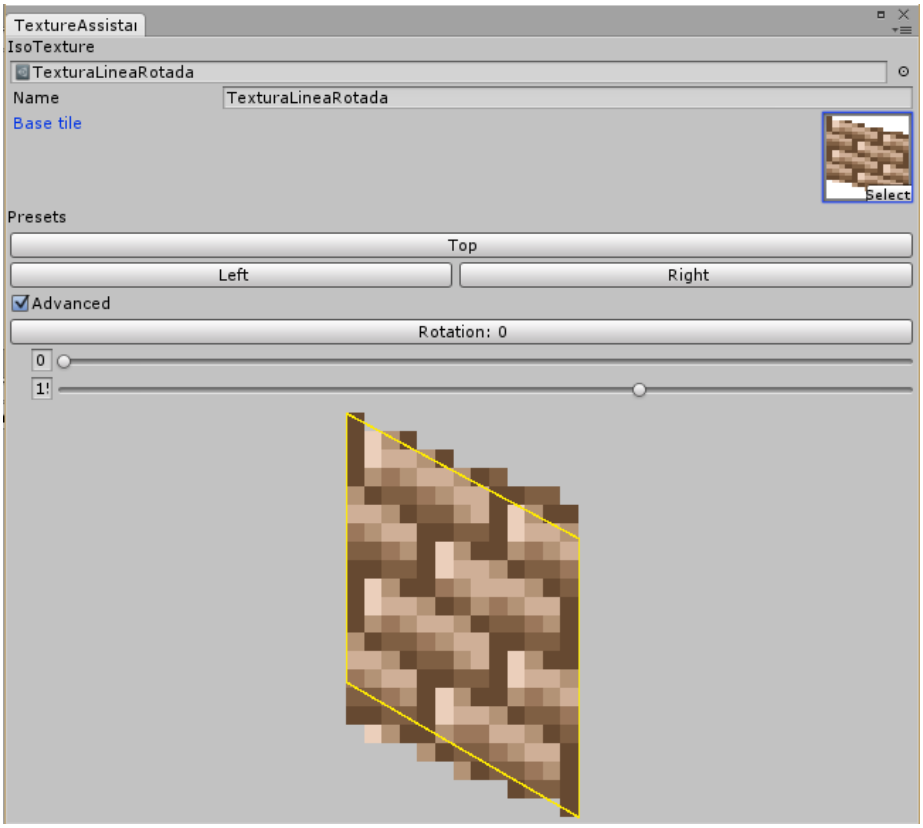


Figura 40 Herramienta Texture Assistant

Pero a la hora de extraer las texturas del juego hay que tener en cuenta varios factores:

- Que la imagen esté en forma de rombo.
- Las dimensiones sean de 32x16 (esto es importante porque en la abadía tradicional son de 32x15, por lo que hay que añadir un pixel, preferiblemente por debajo). Y las de los laterales son de 16x40
- Es importante que en las de la capa superior las esquinas estén en el ([0,16],16) y las laterales estén en (8,[0,32])
- Es importante que cuando se exporte las texturas, sean sólo las texturas, sin código fuente, es decir, asset y png/psd.

Capítulo 5.- *IsoAbbey*: Implementación multiplataforma con *IsoUnity*

En este punto se va a tratar todo aquello relacionado con la realización y el montaje de *IsoAbbey*, cómo se relaciona con *IsoUnity* y los diversos detalles técnicos que hacen posible todo esto.

5.1.- Uso práctico de *IsoUnity* y prueba

En primer lugar se va a explicar el sistema de generación de escenarios, desde la unidad más simple hasta un mapa completo generado, destacando todas las características que proporciona *IsoUnity*. En este punto se recoge la explicación de varios elementos propios del ecosistema nativo como son: "cell" o la unidad básica que compone el mapa, los "decorate" que son aquellos objetos que sobresalen sobre el mapa y lo decoran, los personajes o aquellos individuos que pueblan el juego y el teleporter como sistema que permite la comunicación entre distintos mapas.

A continuación se hace referencia a aquellos elementos que se han añadido a *IsoUnity* durante la elaboración de este trabajo para dotarlo de una funcionalidad real. Se mencionarán brevemente algunos errores o mejoras introducidas que detallaremos en el apartado 5.3.

5.2.- Construcción de escenarios en *IsoUnity*

Para entender cómo funciona la construcción de escenarios en *IsoUnity*, primeramente hay que conocer en que se basa *IsoUnity*. Podemos establecer que la "cell" (Celda) es la unidad básica que lo compone. En la Figura 41 puede verse una.

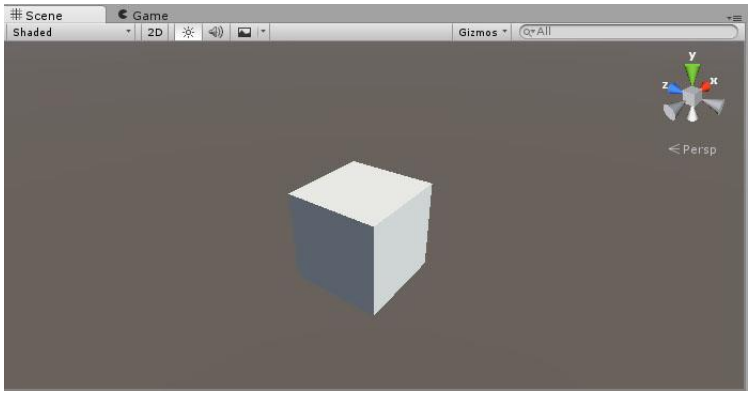


Figura 41 Vista de una cell en el entorno de desarrollo Unity

A la hora de crear un mapa se necesita crear el objeto padre "*IsoUnity* Map" en el cual se irán incluyendo las celdas, con las cuales se crea el mapa. Una "cell" tiene diferentes características físicas que se pueden modificar, estas son la altura y el acabado de la cara superior, todo esto puede verse en la Figura 42.

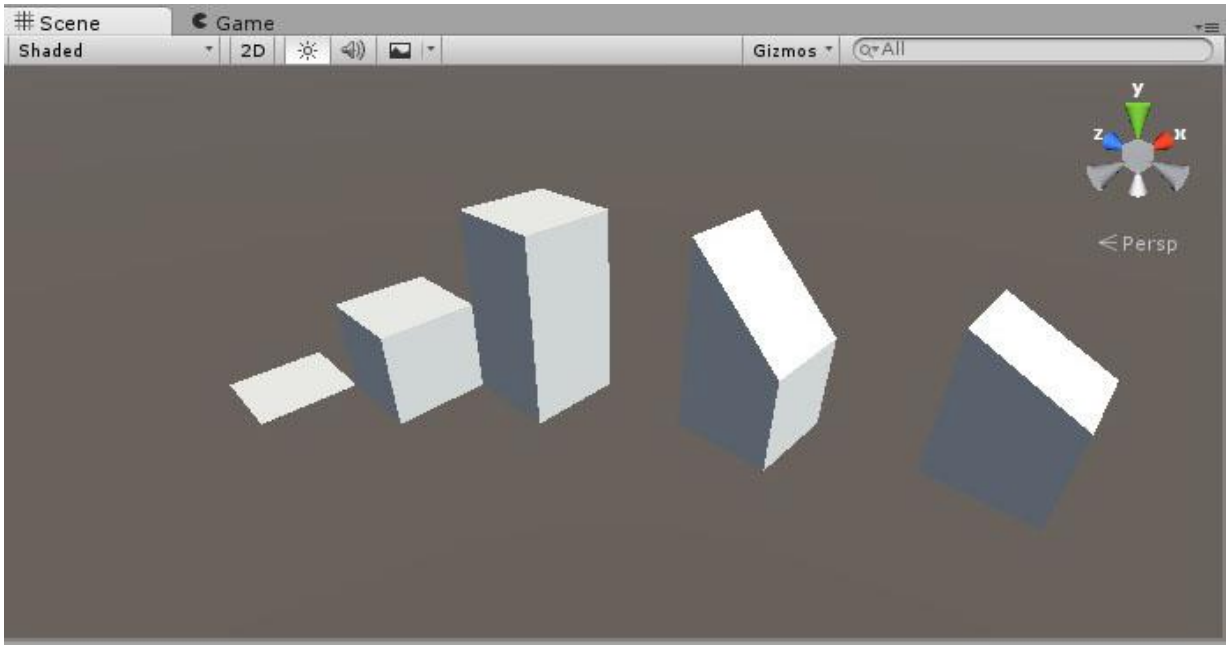


Figura 42 Cells con distintas alturas y acabados superiores

Una "cell" es un GameObject (elemento de Unity que se refiere a un objeto propio) con características de *IsoUnity*. Una "cell" designa una porción cuadrangular del espacio físico con unas medidas fijadas en los ejes X e Y con posibilidades de variar su medida en cualquier momento en el eje Z. Se corresponde con el elemento básico que conforma el mapa, es decir, con aquel espacio por donde el personaje puede caminar (obviando los casos en los que se trate de una pared o cualquier otro elemento diseñado para que no pueda ser transitable). Este elemento incluye un atributo que permite definir si es o no caminable (el atributo walkable), es decir, se puede en tiempo de creación o ejecución gestionar dicha característica, lo que garantiza ampliar los recursos con los que cuenta el diseñador.

Por último las "cell" pueden ser decoradas aplicándolas distintos tintes. Para incluir dichos tintes, el propio *IsoUnity* incluye una herramienta que permite seleccionar la superficie (cara) de la "cell" que queremos pintar y la textura para pintarla (todo esto está explicado con mayor detalle dentro del manual que se proporciona junto con la memoria). Cada "cell" puede tener una altura diferente, como ya se ha visto previamente, y cada uno de estos niveles (incluyendo los medios niveles) puede ser tintado con distintos elementos. Cabe recordar que el "cell" es un GameObject de Unity por lo que se beneficia de los efectos de luces y sombras proporcionados por el motor y gestionados por este. En la Figura 43 se incluyen ejemplos de esto.

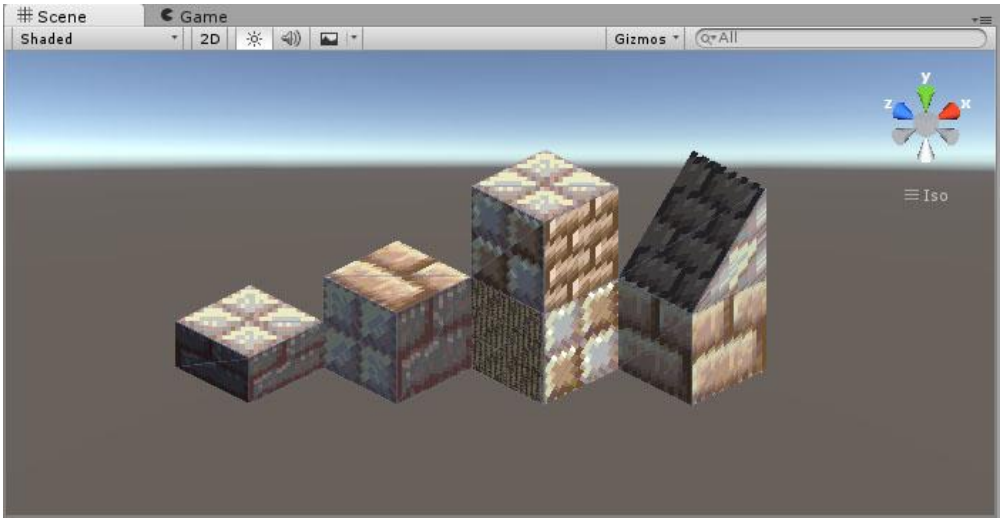


Figura 43 Aplicación de diversas texturas en las Cells anteriores

El siguiente paso es unir varias "cell" para generar una superficie por la cual las entidades así como el protagonista puedan moverse. La altura que puede tener cada "cell" permite gestionar un entorno dinámico formado por varias alturas, replicando con ello escenarios como los que se pueden ver en cualquier videojuego comercial. A modo de apunte, cabe destacar que los gráficos del juego serán tan buenos como los tintes que le conformen, ya que se puede conseguir desde un toque hiperrealista hasta un aspecto de dibujo animado o cómic.

Figura 44 Comparación de un mapa creado en IsoUnity (sin utilizar decoraciones) con un mapa de un videojuego real

Una vez que se tiene la base de un mapa hecho y coloreado, hay que añadir una decoración, esto se hará por medio de los objetos "Decorate", dichos objetos comparten características físicas con las "cell", pero solo se encuentran en un plano 2D. La principal limitación que aporta esto es que se pierde las características 3D que aporta Unity y limitamos el juego a una perspectiva isométrica. Un "Decorate" se coloca sobre una "cell", esto limita el número de elementos que se pueden gestionar a uno por cada "cell".

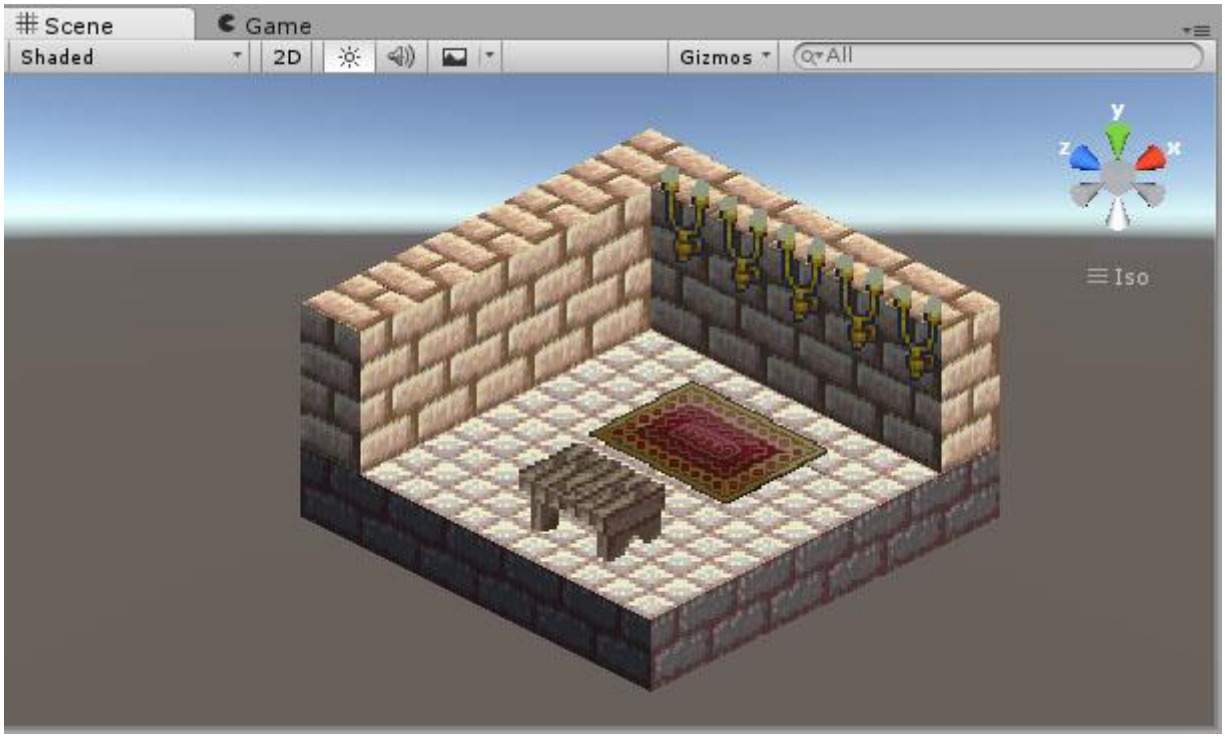


Figura 45 Habitación realizada en IsoUnity que incluye diversas decoraciones

En la Figura 45 pueden verse los 3 tipos de "Decorate" que pueden incluirse en un mapa gracias a *IsoUnity*, la mesa se corresponde con un elemento que se superpone sobre una "cell", pero a efectos prácticos se sitúa sobre varias. Esto implica que se debe tener en cuenta que hay que marcar las "cell" adyacentes como no caminables. El siguiente "Decorate" es la alfombra, como en el caso de la mesa también se coloca sobre una celda y ocupa varias, pero en este caso todas son caminables. Por último, se encuentran las antorchas, un tipo de "Decorate" que se cuelga sobre un "cell", una columna o similar, como se cuelga sobre una columna, la celda adyacente puede ser caminable.

Ahora que el mapa sobre el que se va a desarrollar el juego se encuentra creado y decorado (la decoración no tiene una importancia capital, pero es útil para crear todo el contexto del juego y ayudar a la inmersión) es hora de incluir los personajes. Estos pueden ser de dos tipos: El protagonista y los controlados por el ordenador. Aunque podemos disponer de varios protagonistas en escena al mismo tiempo, debido a las limitaciones de *IsoUnity* (Similares a las de la mayor parte de videojuegos actuales) solo podemos controlar a un protagonista cada vez. Los personajes son un tipo especial de "Decorate" al que se debe añadir elementos proporcionados por *IsoUnity* para distinguirlos de uno de los "Decorate" explicados anteriormente.

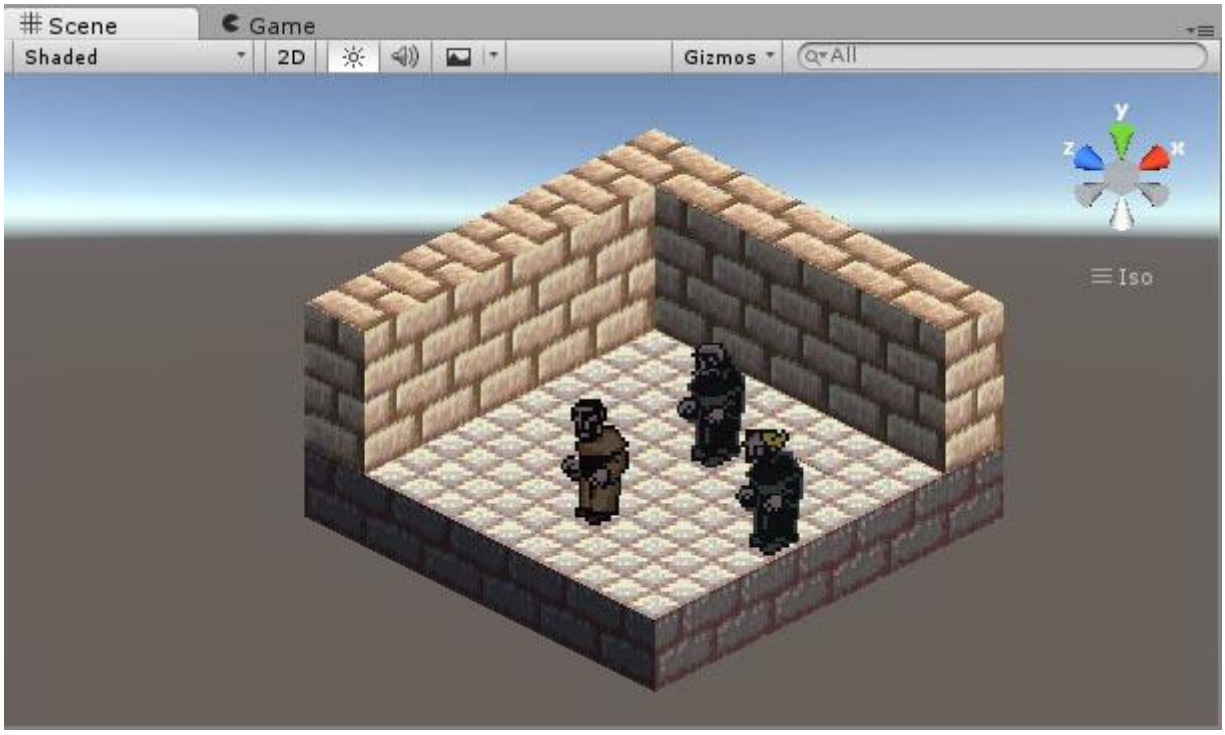


Figura 46 Varios personajes (Entities) en un mismo mapa

A diferencia de un "Decorate" normal, el personaje cuenta con un mayor número de Sprites en función de las direccion (Norte, Sur, Este y Oeste) así como de las posiciones: Paso izq, Paso dcho, stand. Hasta este momento, la herramienta solo se ha probado con humanoides, hay que tener en cuenta que si se utiliza otro tipo de personaje, como un animal, la fluidez del movimiento podría resultar poco natural.

Para poder distinguir entre un protagonista y el resto de personajes, se debe añadir el script "Player". Se ha incluido más información referente a este punto en el manual que se adjunta en los anexos.

El último punto referente a la creación de mapas se corresponde con los "Teleporter". Estos elementos permiten a los personajes moverse entre varios mapas. Debido a como está configurado *IsoUnity*, la cámara del juego se enfoca en un personaje (sea o no el protagonista) y a un espacio relativamente grande a su alrededor que muestra un único mapa. Este elemento permite al personaje moverse entre los distintos mapas creados por la herramienta. En todo momento solo un mapa está activo, de tal forma que en pantalla solo aparece un mapa, esto se puede utilizar para crear habitaciones contiguas al mapa principal o para la gestión de estancias independientes, lo que libera de recursos a la máquina al cargar mapas más pequeños y dota de gran libertad al creador.

Se puede encontrar más información sobre menús y funcionalidades así como capturas del proceso en el manual que se proporciona junto con la memoria. En dicho manual también se incluye la información necesaria sobre los scripts que se tienen que añadir a cada uno de los elementos.

5.3.- Características nuevas

Cabe recordar que *IsoAbbey* está implementado sobre *IsoUnity*, quien se asiente en Unity de la siguiente forma (Figura 47).

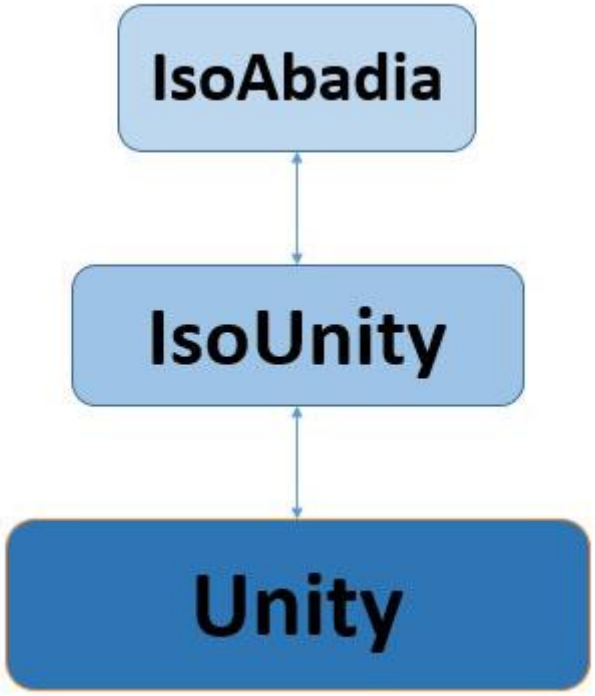


Figura 47 Esquema de arquitectura empleada

Es importante destacar esto en primer lugar ya que el proyecto de *IsoAbbey* está implementado sobre *IsoUnity* y se puede considerar que es paralelo a él, ya que cuando se actualiza éste se puede traer dicha actualización sin hacer, en principio, ningún cambio. Se sigue el mismo proceso cuando salen nuevas versiones de Unity.

5.4.- Elaboración del control del juego

El desarrollo de *IsoAbbey* está basado en dos partes: Primeramente se ha replicado el día 1 de *La Abadía del Crimen* y en segundo lugar se ha dejado una especie de sistema de exploración en el que se puede observar cómo transcurre el día a día en una abadía medieval.

5.4.1.- Día 1 en *La Abadía del Crimen*

Este proyecto ha tratado de, en primer lugar, replicar el Día 1 de *La Abadía del Crimen*. A continuación se expone el resumen de dicho día, es decir, que ocurre en cada momento y la arquitectura que ha permitido traer esto a *IsoUnity*.

En el juego original, tras darle al botón de jugar, se muestra una especie de prólogo en el que se le presenta al jugador una pequeña historia de la abadía y los hechos que han llevado a nuestros protagonistas a dirigirse allí. Esto se hace por medio de una ventana en la que se va presentando el texto. Esto puede verse en la Figura 48.

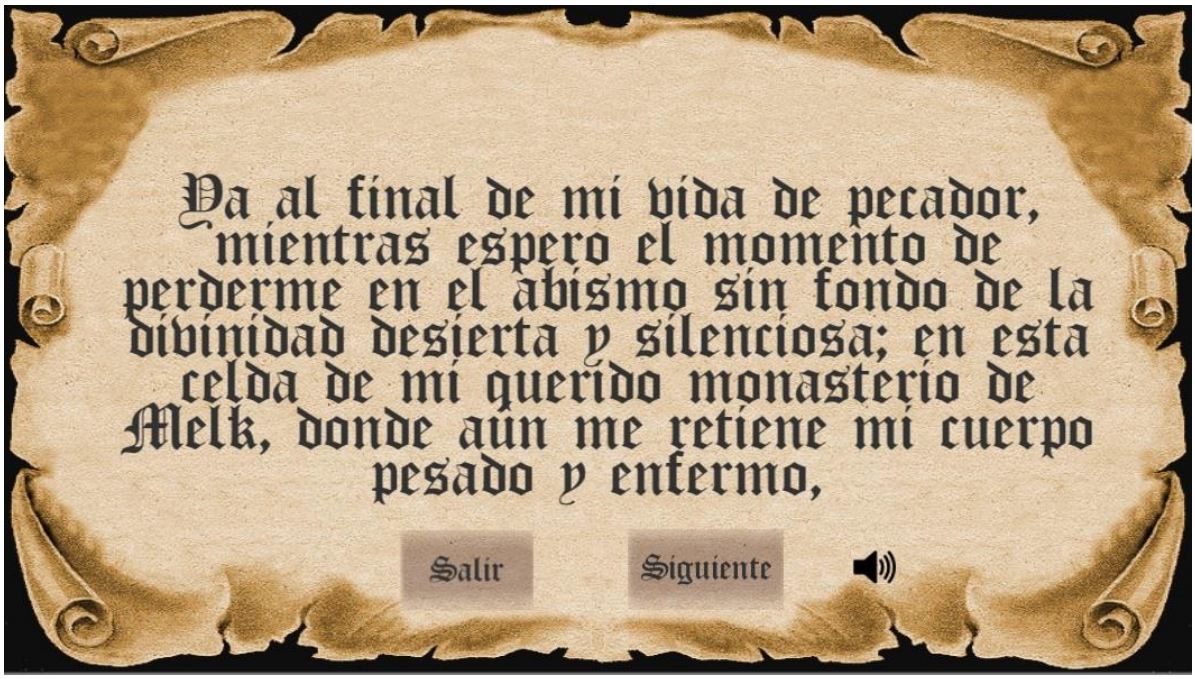


Figura 48 Prólogo del videojuego

A continuación aparece la abadía con una imagen de Guillermo y Adso, donde se puede mover a ambos en una única dirección (el único camino posible es en dirección norte) "obligando" al jugador a encontrarse con el abad, quien introducirá al jugador en las normas de la abadía y nos insta a que le sigamos, haciéndole perder a Guillermo en caso contrario Obsequium (el principal recurso del juego, la vida).

El abad guía al jugador por una parte de la abadía hasta la celda de Guillermo. Una vez allí, le deja un tiempo libre de investigación de la abadía hasta llegar a la hora de la misa, estando obligados a ir como se menciona en las normas que nos dice al principio. Una vez en la misa, se muestra al jugador la imagen de otros monjes llegando a ella. Tras finalizar el periodo de oración, el abad nos acompaña hasta nuestra celda donde al llegar Adso nos insta a dormir, dando por finalizado el día 1.

Con esto termino el análisis de ese primer día y todo lo que se va a trasladar a *IsoUnity*. Se observó que ciertas características de esta versión no tendrían demasiado éxito hoy

en día. Esto es debido a que los juegos tenían que ajustarse a las limitaciones técnicas de la época y por esto se buscaba elevar el nivel de dificultad para tratar de alargar la duración del videojuego. Por tanto se ha tratado de hacer este remake manteniéndose lo más ajustado posible al original pero modificando aquellas áreas que podría entrar en conflicto con la jugabilidad de nuestros tiempos.

Para orquestar los sucesos de la trama del juego se ha utilizado una máquina de estados, a continuación se muestra como se ha implementado:

```
public class prMaquinaEstados
{
        protected List<pEstados> m_estados;
        protected int m_estadoActual = 0;
        protected int m_estadoAnterior = -1;
        protected ControladorPresentacion m_controlador;
        public prMaquinaEstados(ControladorPresentacion controlador)
        {
                m_controlador = controlador;
                m_estados = new List<pEstados> ();
        }

        public void EjecutarMaquina()
        {
                m_estados [m_estadoActual].Ejecutar ();
                int prev = m_estadoActual;
                m_estadoActual = m_estados [m_estadoActual].Transicion
(m_estadoActual, m_estadoAnterior);
                if (m_estadoActual != prev)
                        m_estadoAnterior = prev;
        }
}
```

El código anterior se corresponde con la definición que se ha dado a la máquina de estados. Está compuesta por una lista donde se van agregando los distintos estados que la conforman así como dos enteros que hacen referencia a los estados Actual y Anterior. El estado Anterior hace referencia al estado que ha invocado al actual, es decir, puede ocurrir por ejemplo que estado anterior sea el 7 mientras que el actual sea el 24. Además de la constructora la máquina incluye una única función que se está ejecutando de forma continua.

A continuación se muestra lo que es un estado:

```
public abstract class pEstados
{
        protected ControladorPresentacion m_controlador;

        public pEstados(ControladorPresentacion controlador)
        {
                m_controlador = controlador;
        }
        public abstract int Transicion(int state,int previousState);

        public abstract void Ejecutar();
}
```

Como se puede observar un estado mantiene una referencia a la clase general donde se encuentran los distintos elementos propios de la escena actual. La clase se compone de dos funciones Transición y Ejecutar. Ejecutar contiene la definición del estado actual del mundo, es decir, controla ciertas acciones que están ocurriendo actualmente en pantalla. Se controlan ciertas acciones ya que para la gestión de la reconstrucción se han utilizado varias máquinas que trabajan al mismo tiempo para poder gestionar todos los requisitos definidos previamente. Por otro lado la función transición suele estar controlado por un sistema de "ifs" que trata con las distintas opciones de estados a las que se puede ir desde el estado actual.

5.4.1.1.- Máquina de estados controladora de la trama principal

Este punto se encarga de mostrar la arquitectura con la que se ha desarrollado el remake. A continuación se muestra una imagen que contiene la relación de estados de la máquina que se refiere al día 1 con la interacción de Guillermo, Adso y el Abad:

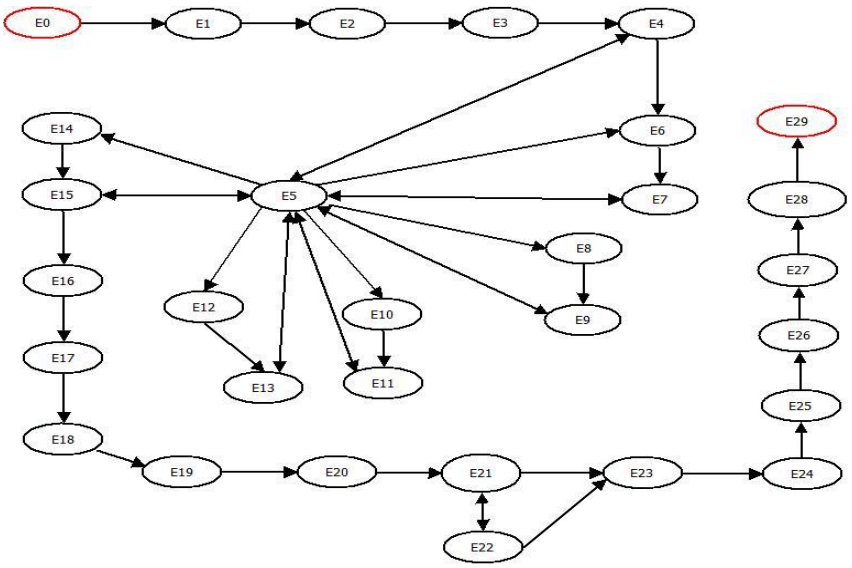


Figura 49 Estados pertenecientes a la máquina principal

Estado 0: Muestra el cartel del “día 1” y el sonido que le acompaña, pese a no tener relación con los personajes anteriormente especificados se ha decidido incluir aquí ya que ésta es la máquina principal que controla la trama en este día.

Estado 1: El jugador permanece en este estado mientras se familiariza con los controles y termina cuando llega delante de la celda del abad.

Estado 2: Durante este estado se mostrará un cartel con el recibimiento que el abad da a los protagonistas. Esta es una de las principales modificaciones de diseño realizadas frente al videojuego antiguo, ya que lo que antes se hacía en un pequeño rectángulo dentro del HUD ahora se ha desligado de éste mostrándose ahora en un cartel. Tras un corto período de tiempo, se oculta dicho cartel y el abad anuncia que va a moverse instando al jugador a seguirle.

Estado 3: Este estado controla únicamente el movimiento del abad a la nueva celda, esto es debido a una decisión de diseño, ya que en un primer momento los únicos movimientos que podían controlarse eran los relacionados por el jugador (al pulsar teclas) o que tenían como resultado una interacción directa del jugador con otro personaje. Se han implementado funciones que permiten mover personajes de forma independiente a la interacción del jugador.

Estado 4: Este estado se encarga de medir la distancia que hay entre el jugador y el abad, es decir, que el jugador haya seguido al abad y muestra un cartel de advertencia que insta a seguirle. Hay dos posibles salidas a esta situación en función de la distancia medida anteriormente. En caso de que la distancia sea mayor a la estimada por el diseño del juego se avanza al estado 5, en caso contrario al 6.

Estado 5: Este estado contiene interacciones con una gran cantidad de estados de esta máquina ya que hace uso de la utilidad estado anterior para regresar al estado que le invocó en caso de no cumplir con las condiciones necesarias o avanza al [estado anterior + 1] lo que provoca que se avance en la trama. La otra función de este estado es la de restar puntos de vida al jugador en tanto en cuanto no se cumplen los objetivos de la trama, aunque esto se realiza de forma más laxa que en la versión original.

Estado 6: El abad muestra nueva información sobre los hechos de la abadía y se desplaza a una nueva celda.

Estado 7: Vuelve a comprobar la distancia a la que se encuentra el jugador para en caso necesario ir al estado 5.
Aclaración: En varios puntos de la trama aparecen pares de estados ligados, como es el caso del estado 6 y 7, en los cuales en uno se avanza y en otro se comprueba la distancia con el jugador. Se fomenta esta elección de diseño ya que permite la realización de cambios y modificaciones en el modelo que afectan a hechos concretos. El otro diseño posible estaba basado en unificar todos estos estados en uno solo y por medio de variables de control realizar una acción u otra, característica que se acaba desestimando

ya que se buscaba una implementación más simple que permitiese modificaciones futuras.

Estados 8 y 9: Avance del abad con nueva muestra de la historia y control de la distancia. Otra característica de estos estados que afianzan la laxitud frente a la abadía original es que existen un "delay" entre el movimiento del abad, que es automático, y el del jugador, que puede sufrir distracciones (lectura del texto, no sigue la ruta más óptima…) es por esto que en función de la distancia entre las celdas se asigna un tiempo de espera antes de que se compruebe la distancia.

Estados 10 y 11: Otro par de estados anidados con movimiento y distancia, en estos estados no se muestra texto de información referente a la historia de la abadía.

Estados 12 y 13: Estados anidados que vuelven a mostrar información sobre la historia.

Estados 14 y 15: El abad nos deja delante de nuestra habitación y con este estado terminan los movimientos hasta este punto del día 1, que podría considerarse como la mitad del primer día.

Estado 16: El abad indica al jugador que ésta será su habitación y se marcha, devolviendo en este punto la libertad de acción al jugador.

Estado 17: Adso informa al jugador de que dispone de tiempo libre para investigar la abadía, está estimado que sean 3 minutos pero debido a una modificación necesaria para la presentación del TFG se ha rebajado a unos pocos segundos.

Estado 18: En este estado hay un contador que regula el control de "exploración" antes de seguir con el resto de la trama.

Estado 19: Con el tiempo de "exploración" concluido, Adso informa al jugador que debe acudir a misa.

Estado 20: Adso guía al abad hasta la iglesia desde la posición en la que se encuentran en ese instante.

Estado 21: Con Adso ya en la iglesia, se comprueba si el jugador le ha seguido. En caso afirmativo, avanzamos al estado 23 y en caso contrario se pasa al estado 22.

Estado 22: Se encarga de mostrar el mensaje al jugador y restar puntos de vida.

Estado 23: Mientras el resto de monjes van llegando a la misa, se quita el control de Guillermo al jugador y se selecciona a un nuevo monje no controlable, fijando la vista en él.

Estado 24: Ahora el jugador solo puede observar como el monje se dirige hasta una puerta que cierra.

Estado 25: El jugador ve como dicho monje acude a la misa y ocupa su posición.

Estado 26: Se devuelve la cámara al jugador pero no el control del juego.

Estado 27: El abad comienza la misa al tiempo que se muestra un nuevo cartel con información que da paso a una cortina en negro a modo de cambio de escena.

Estado 28: Tras acabar la misa, se muestra un mensaje que informa sobre la necesidad de los monjes de acudir a sus celdas donde deberán pasar la noche. Al ser el primer día el abad nos acompañará a la nuestra.

Estado 29: Al llegar a nuestra celda, se muestra un mensaje al jugador dándole la opción de entrar a dormir o continuar, donde la única respuesta aceptada por la trama del juego es dormir.

5.4.1.2.- Máquina de estados controladora del resto de monjes

La máquina anterior controla la narración del juego, es decir, la historia principal planteada por el diseñador. Por otra parte el resto de personajes no importantes en dicha trama o que no tienen papel en la parte actualmente controlada, son ubicados en la otra máquina.

Esta nueva máquina ha recibido el nombre de “máquina de monjes” y es el término con el que la nombraremos a partir ahora. La Máquina de monjes tiene un estado por cada uno de las horas canónicas y su configuración depende del día en el que estamos, porque ciertos monjes no tienen papel en el día uno pero si en otros y hay monjes que desaparecen de la trama porque van muriendo.

Como ya se ha explicado anteriormente las horas canónicas son: Oficio, Laudes, Tercia, Sexta, Nona, Vísperas y Completas. En los anexos se puede encontrar un análisis detallado por cada monje en cada una de las horas. Dicho lo cual encontramos 7 estados con condiciones que varían por cada día en la función de ejecución, mientras que la función de transición se activa con el cambio de la hora, esto es controlado por la máquina de la trama.

Por simplificación en los estados de transición de esta máquina se ha decidido incluir la posibilidad de saltar a cualquier estado, ya que esto garantiza posibles cambios futuros en la trama. La parte de ejecución tiene una complejidad algo mayor, debido a que ha tenido que hacerse un "apartado" para cada uno de los monjes y todos estos puede que tengan una configuración diferente para según qué día. A continuación puede verse la estructura:

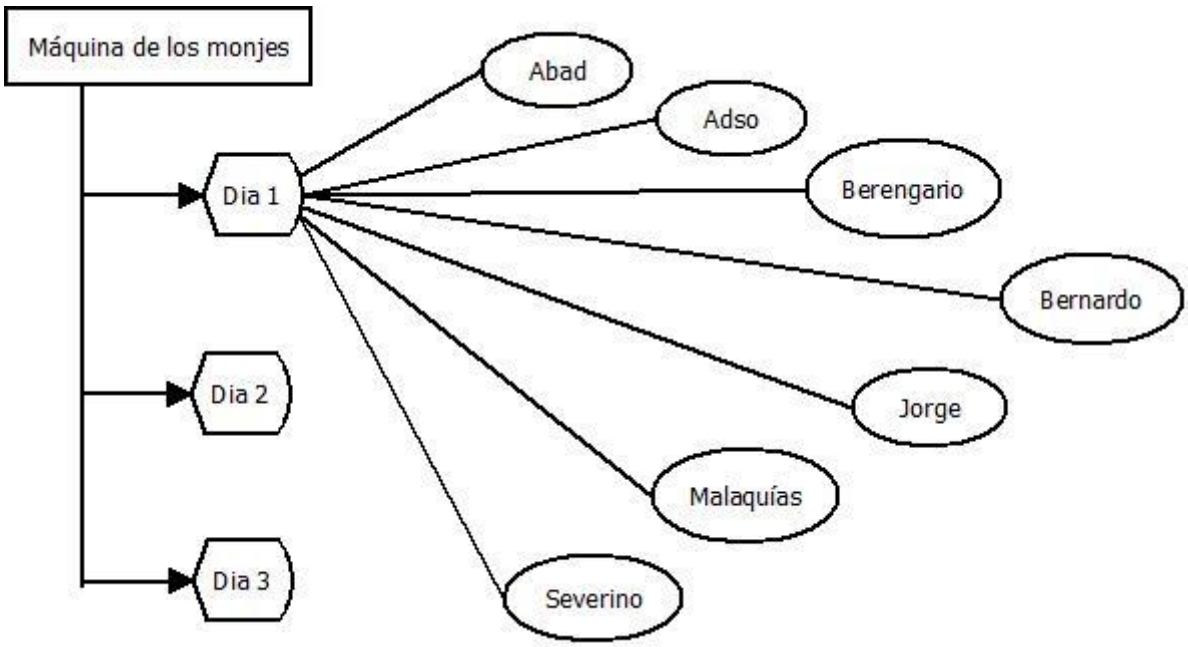


Figura 50 Estados pertenecientes a la máquina de los monjes

Se optó esta solución aun sabiendo que no era demasiado óptima debido a que era la menos mala de las opciones que se barajaban. Existe otra opción que empezó a diseñarse aunque finalmente se abandonó en favor de esta. Se basaba en diseñar una máquina para cada monje (7 máquinas potenciales) pero al estar ejecutándose en todo momento debido a *IsoUnity*, funcionan de forma similar a la función *update*, es decir, se ejecutan por cada tick de reloj. Pero esto demostró ser poco eficiente en cuanto al nivel de fps ya que debido a su escasa optimización producen grandes caídas en los fps, por contra la opción tomada resultó ser más óptima.

5.4.2.- Elementos implementados necesarios para *IsoAbbey*

IsoUnity es un plugin desarrollado para la creación de escenarios y gestión básica de personajes que incluye además otra serie de funcionalidades ya desarrolladas anteriormente y que puede ser comprobada en el manual de *IsoUnity* incluido en los anexos. Debido a esto es necesario crear toda una serie de elementos adicionales para poder crear el juego deseado. Se puede entender *IsoUnity* como una base sobre la que desarrollar un videojuego. A continuación se explican los módulos o funcionalidades que se han añadido.

Para el desarrollo de estos elementos se observaron GamePlays del videojuego original y se jugó para poder extraer todas las características necesarias para el videojuego.

5.4.2.1.- Funciones de movilidad

El movimiento del personaje principal es controlado por *IsoUnity*, el resto de personajes pueden moverse gracias a interacciones específicas con el protagonista, por ejemplo el protagonista se acerca a un NPC y activa un evento mediante el cual dicho NPC se mueve hasta otra ubicación.

Se ha desarrollado una función que permite mover a los personajes libremente. Los elementos que utiliza son: La Entity (elemento de *IsoUnity*, todos los "personajes" del juego son Entitys) que es quien se mueve y la Cell, que es a donde se mueve. A partir de esto se desarrolló la siguiente característica, esta es el seguimiento de personajes.

Figura 51 Función Movimiento libre

Para que una Entity puede seguir a otra hay que combinar dos elementos: El movimiento y la distancia. La distancia es una función desarrollada que mide la lejanía entre dos elementos seleccionados, se llama a dicha función incluyendo en la llamada las Entity. A partir de aquí la función nos devuelve un float con la distancia, dicho float puede ser utilizado en un script que una estos dos elementos de tal forma que cuando el personaje perseguido se aleja del perseguidor, éste se mueve hasta la casilla adyacente del perseguido.

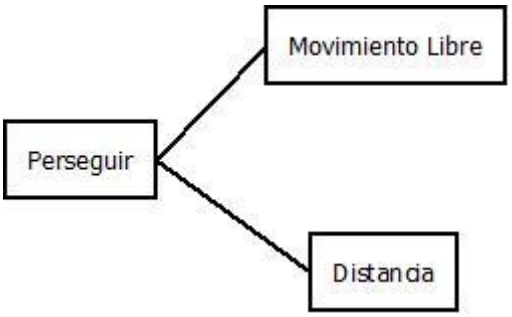


Figura 52 Función perseguir

Otra característica necesaria para poder hacer la reconstrucción es lo que se define como "las patrullas". En determinadas características (dias y horas) ciertos monjes realizan movimientos rutinarios. Para poder recrear dichos movimientos se estableció la siguiente solución: A partir del movimiento libre puede controlarse que el personaje se mueva de una celda a otra, si se une esto con el elemento de Unity BoxCollider, el cual permite lanzar eventos cuando dos objetos colisionan se puede conseguir un movimiento perpetuo. La complejidad de esta patrulla se basa únicamente en el número

de celdas de control introducidas, logrando con esto el movimiento deseado. Se compone un script con esto y su activación se controla desde alguna de las máquinas de estados.

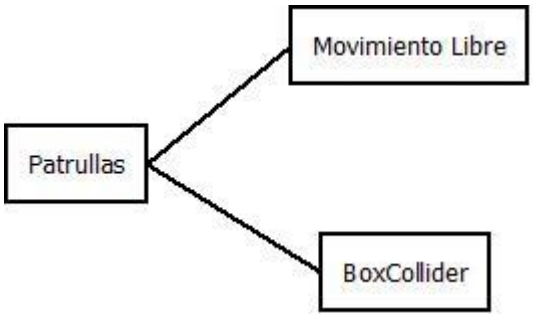


Figura 53 Función patrullar

El juego original contiene una característica que no ha podido ser emulada, esto es que se podía mover a Guillermo media casilla. Debido a una limitación técnica en la arquitectura de *IsoUnity* esto no ha podido ser replicado (realmente no es una "limitación", sino que es así como está construido)

5.4.2.2.- El HUD

El HUD es la información que se muestra en pantalla para el jugador, ésta información no se encuentra presente en el mundo del videojuego, sino que sirve como puente para conectar al jugador con el juego permitiendo que este vea todos los elementos necesarios que garanticen la máxima jugabilidad.

Tras observar el HUD original se determinó que este era muy arcaico (propio de la época) por lo que se optó por un rediseño tratando de mantener elementos en medida de lo posible como guiño al videojuego original. A continuación se puede ver una comparación entre *IsoAbbey* y *La Abadía del Crimen* (el remake).

Figura 54 Comparación de Hud entre nuestro juego y el original

En líneas generales el HUD presenta un aspecto conservador, pero ahora ha dejado de ser un objeto sólido que ocupa la parte inferior del espacio de juego y ha adaptado un aspecto más dinámico. Se ha desarrollado utilizando el Interfaz proporcionado con Unity utilizando botones e imágenes.

El sistema de vida se ha desarrollado como una clase controlada por la máquina de estados principal. Si se observa dicha máquina se puede ver en el estado 5 que se resta un punto de vida al llegar a él. El videojuego original no permite recuperar puntos de Obsequium por lo tanto se ha implementado únicamente una función que disminuye la vida. Esto lo hace restando los puntos de vida internos del personaje y modificando la barra.

Se ha conservado el sistema de días y horas del juego original, tanto el entero como el string con la hora se cambian desde la máquina principal. El cambio de estado en la máquina de los monjes depende del contenido actual del espacio horas.

Si se mira la imagen de la de derecha, puede verse un espacio en negro, que es donde aparece el texto proveniente de la conversación con los distintos personajes. En la versión actual se ha decidido sacar el texto de la barra inferior y se ha situado en una franja superior izquierda. Esto puede verse en la siguiente imagen:

Figura 55 Imagen de los carteles mostrados en el juego

El siguiente punto a destacar es todo lo relacionado con el control de personajes. En el juego original el personaje se movía eligiendo la dirección y después pulsando avanzar. Para actualizar esto a una versión algo más actual se permitió controlar el juego con WSAD, con un pad, o con un touchPad (tablets y móviles). Además de esto antes podía controlarse, de forma muy limitada a Adso, para cumplir ciertos objetivos. Esto ha sido completamente reconstruido.

En esta nueva versión ambos personajes pueden moverse utilizando todos los controles (esto no ocurría originalmente). Esto conduce a un nuevo requisito, el control de la cámara. *IsoUnity* no proporciona control sobre la cámara, al crear un mapa se designa un Entity al que enfocar. Para solucionar esto se desarrolló un script que permitía desde la máquina principal (ciertos momentos del juego requieren que se centre la atención sobre otros personajes) o desde el Interfaz.

Los retratos situados a ambos lados pertenecen a Guillermo y Adso, dichos retratos son botones seleccionables que muestran a color la imagen del personaje que podemos controlar y una pequeña franja en colores verde o rojo que señalan si está activa la

función de seguimiento. Esto permite realizar ciertos puzzles que añaden variedad y dinamismo al juego, ya no se basa únicamente en ir a sitios (o estar en ellos) para cumplir objetivos.

El último elemento del HUD es el sistema de inventario. Se puede observar que mantiene un aspecto similar al de la versión original, pero ahora se puede pulsar sobre los objetos para obtener una descripción de los mismos (Esto no aparece en la versión original). Los objetos pueden recogerse por dos vías: Pasando sobre ellos (funcionalidad añadida y mostrada en el vídeo de presentación) o al interactuar con ellos (funcionalidad de *IsoUnity*) cuando no se puede caminar, por ejemplo cuando están situados sobre una mesa.

5.4.2.3.- Otras funcionalidades desarrolladas

Por último se recogen en este apartado las dos últimas características desarrolladas: Un módulo que permite gestionar los sonidos que se van reproduciendo y el módulo conversacional. Ambos modelos están en una fase poco avanzada aunque funcional ya que dependen en gran medida de *IsoUnity*.

IsoUnity está siendo actualizado constantemente y aún se encuentra en una fase inicial que presenta ciertos bugs y elementos que pueden ser optimizados (sobre esto se incidirá en el capítulo 6). Por diversas eventualidades se decidió desarrollar un elemento que permitiese gestionar las conversaciones. Actualmente *IsoUnity* incluye un sistema de eventos, a partir de los cuales se pueden gestionar actividades como: Iniciar conversaciones y lanzar movimientos. El sistema de creación de dichos eventos es muy arcaico (no ha sido actualizado desde la primera versión) y durante la creación de los primeros eventos para *IsoAbbey* presentaba diferentes problemas, por esta razón se procedió a la creación de un módulo paralelo.

El módulo desarrollado está basado en dos componentes básicos: Un script de control y unos elementos de interfaz. Los elementos del interfaz (que pueden verse en el vídeo) están formados por un fondo básico, una imagen de interfaz, una imagen del personaje que no está hablando y un espacio para el texto. Estos elementos son controlados desde el script, que gestiona los textos que se van mostrando así como cuando han de aparecer cada una de las ventanas, así como la imagen del personaje. Por último se incluyen varias variables de control que han de ser activadas mediante botones para poder avanzar en la conversación. Cada uno de los botones ha de ser personalizado para cada conversación, pero esto se hizo de forma provisional mientras se esperaba a que *IsoUnity* actualizase dicha *feature*.

Por otra parte se encuentra el módulo de sonidos. Había que desarrollar un sistema que permitiese gestionar todos los sonidos que tuviesen que reproducirse de forma que cada vez que se necesite reproducir alguno se llame a este módulo y sea él quien decida el sonido que procede. Esto funciona para elementos como la recogida de items, los sonidos de fondo o el sonido que se produce al mostrar el cartel de “dia X”. Pero falla

con sonidos como el movimiento de pasos, ya que no se puede distinguir entre quien realiza el movimiento (debido a la versión actual de *IsoUnity*).

Estos elementos cuentan con características que pueden considerarse temporales y que es posible que en versiones futuras de *IsoUnity* tengan que sustituirse completamente por su versión correspondiente o actualizarse para permitir su completa integración.

Capítulo 6.- Discusión sobre *IsoAbbey* y el futuro de *IsoUnity*

En este punto se va a desarrollar toda la problemática asociada a trabajar con una herramienta nueva, las soluciones encontradas y las mejoras que se han producido en dicha herramienta. Por último se darán los resultados de la forma más objetiva posible de cómo se ha desenvuelto *IsoUnity* al ser sometido ante un entorno de producción y de si es posible un proyecto como el remake de *La Abadía del Crimen* para ser abordado por alguna compañía desarrolladora de videojuegos.

Como ya se ha mencionado en capítulos anteriores, gran parte de esta información puede verse online en la plataforma *GitHub*, que es aquella que se utilizó como repositorio de información así como para dar visibilidad al proyecto y poder recibir *feedback* por parte de usuarios experimentados.

6.1.- Problemas de usabilidad de *IsoUnity*

Lo primero que hay que destacar es que *IsoUnity* no disponía de ningún tipo de manual ni de instalación ni de uso, por lo que podría calificarse su aproximación a él como ardua. La solución a este problema fue contactar con los desarrolladores y mediante de varias reuniones, tratar de establecer los conocimientos necesarios para crear un primer mapa e introducir algún personaje que pudiera moverse. Se decidió desarrollar un manual sobre la herramienta para que los próximos usuarios no tuvieran el mismo problema, por lo que consideramos que este problema ha resultado completamente solucionado.

En cuanto a la creación de escenarios, se encontraron varios problemas relativos a la usabilidad. El primer problema es relativo al pintado de la "Cell" así como del tamaño de la ventana. Actualmente se pueden pintar un plano de "Cell" con un solo click y seleccionar la opción de elegir entre simple o múltiple. El otro problema relativo al pintado estaba basado en el tamaño de la ventana de selección del tinte, es decir, antes había que hacer scroll debido a que únicamente aparecía una fila de elementos a seleccionar. En la actualidad el tamaño de la ventana permite la selección de tintes en varias filas sin tener que clicar sobre el scroll, lo que agiliza mucho la operación.

Otro problema relativo al pintado de Cell es la gran acumulación de tintes en proyectos relativamente grandes. No se dispone de la posibilidad del uso de carpetas para la organización, por lo que un proyecto que utilice más de 50 tintes puede encontrarse con un problema de organización y cualquier proyecto debería utilizar una cantidad de tintes mucho mayor.

La ventana de decoraciones presentaba los mismos problemas que la de texturas y la consideración sobre la creación de carpetas también es extensible a este punto.

Hasta la incorporación en las versiones posteriores de prefabs (objectos prealmacenados en Unity que cuentan con todos los elementos en pantalla para ser utilizados) la creación de mapas era algo engorroso y poco práctico. Actualmente y debido a varias mejoras, este proceso se ha agilizado enormemente, haciendo de la creación de mapas algo mucho más intuitivo.

6.2.- Bugs encontrados en el desarrollo

A continuación se encuentra una lista de todos aquellos problemas encontrados hayan sido solucionados o no, éstos pueden encontrarse en el GitHub del proyecto.

- El bucle principal del juego está buscando en todas las entidades que realizan cualquier acción en todo momento. Esto ha de cambiarse para que las entidades que tengan que realizar alguna acción se tengan que suscribir a dicho bucle.

Éste fue uno de los primeros problemas encontrados y que producía que en un mapa relativamente grande con numerosas entidades sufriera caídas en los fps en tiempo de ejecución.

- Existe un problema relativo al tamaño de las decoraciones, éstas son demasiado grandes.

Se encontró un error relativo al tamaño de las decoraciones, éstas eran demasiado grandes debido a que no obtenían correctamente el escalado. *IsoUnity* funciona con proporciones debido al escalado que sufren en el apartado IsoSettings (un elemento que hay que cumplimentar antes de empezar a hacer algún mapa, en él se contienen los elementos pertenecientes al tamaño de la textura Cell, a partir de la cual se genera el tamaño del resto de elementos). Se localizó la línea de código relativa a dicho problema y se comunicó a los desarrolladores para subsanar dicho error.

- Problema del fijado de la decoración.

Este problema fue producto del error anterior, una vez se solventó el escalado se originó un nuevo problema. Dicho problema producía que al seleccionar una decoración y deslizar el ratón para colocarla en el sitio que desease el desarrollador, se iba produciendo un reescalado en cada una de las casillas. Dicho reescalado provocaba que se crease una copia de la decoración en cada una de esas casillas.

- Serialización de objetos innecesaria.

En un primer momento *IsoUnity* serializaba todos los elementos del mapa. *IsoUnity* funciona de forma que las Cell se establecen como una malla. La primera celda mantiene una referencia a la malla compuesta por una celda, la segunda celda contiene la referencia a la malla y por tanto a la primera celda que también mantiene una referencia a la malla… Esto produjo un problema en espacio que había que arreglar antes de poder pasar a cualquier otro punto.

En Noviembre se esbozó un primer mapa de la abadía que estaba formada por unas 7000 celdas y que pesaba en memoria cerca de dos gigas. Esto era intratable, ya que el mapa se cargaba en Memoria Ram y cada vez que el protagonista se movía actualizaba su posición en la malla, por lo que había que actualizar la posición en todas las celdas…

Se solucionó este problema localizando los elementos que se serializaban innecesariamente y eliminando dichas serializaciones. Actualmente el mapa de un tamaño similar ocupa un espacio en memoria inferior a 200 megas, lo que consideramos como un completo éxito.

- Actualización de *IsoUnity* a Unity 5.

Esto estaba considerado como uno de los objetivos del TFG. Cuando se comenzó en Octubre, estaba anunciado que iba a salir una nueva versión de Unity y el coordinador solicitó que *IsoUnity* debería encontrarse en dicha versión. Esto no originó demasiados problemas, ya que las herramientas proporcionadas por Unity permitían actualizar el código a la versión actual. Como resultado del proceso únicamente algunas partes del código tuvieron que ser adaptadas y actualmente *IsoUnity* se encuentra completamente funcional en la versión más actual de Unity.

- Problemas relativos al movimiento con el ratón.

El movimiento con el ratón presentaba problemas diversos. No funcionaba con todos los clicks y en algunas ocasiones había que pulsar varias veces para que el personaje pudiera responder al movimiento. Dicho problema fue solucionado por los desarrolladores de *IsoUnity*.

- Nuevos problemas con las IsoSettings, éstas pierden la configuración actual o no detectan cuando su configuración es errónea.

Este error se soluciona comprobando el tamaño de la imagen de muestra introducida, es decir, que si dicha imagen no mantiene ciertas proporciones es descartada.

- Poder parar un movimiento.

Cuando se hace click sobre una celda lejana y a continuación se pulsa sobre otra celda, el movimiento hasta esta segunda celda no se produce hasta que la entidad que se mueva no llega a la primera. Esto se producía debido a que los movimientos se colocaban en una pila y se iban resolviendo en orden FiFo. La solución a esto paso por modificar la celda destino al apilar nuevos movimientos debido a que el algoritmo A* (el cual controla el movimiento) se comprueba en cada celda.

- Creación de listas de GameEvent para poder permitir sistemas conversacionales complejos.

Anteriormente no se podían establecer varios GameEvent, por lo que las acciones preprogramadas ocurrían de una en una, lo que producía que si se quería establecer una conversación por cada respuesta dada había que volver a hacer click para poder iniciar el siguiente evento de la conversación. Actualmente se dispone de una lista de

GameEvent, pero debido a que se encuentra aún en una versión algo arcaica se decidió desarrollar un módulo paralelo que realiza un trabajo semejante.

- El problema de la serialización originó un nuevo error del que no fuimos conscientes hasta que no se cambió de escena. Al cambiar de escena se perdían todas las texturas aplicadas al mapa de la escena anterior.

Esto era debido a que cuando se eliminaron los elementos de serialización no se pensó que había que guardar las texturas ya que estas se mantenían en pantalla. La solución pasó por serializar una única malla con las texturas relativas a las celdas.

- Varios objetos con el componente Player (que indica quien es el jugador y se mueven en función a las teclas WSAD, ratón…) responden a las acciones aunque solo uno de ellos tenga dicho componente activado.

Debido al remake se necesitaba que en ciertas ocasiones cambiase el personaje que se controlaba aunque se desactivase el componente player. En un primer momento se optó por crear un sistema de semáforos que controlaban que elemento contaba con el componente player actualmente activado, mientras se esperaba a que los desarrolladores arreglasen dicho problema. Actualmente este error está arreglado.

- Los personajes no se giran hacia una celda cuando no pueden moverse a dicha celda.

Cuando un personaje colocado junto a una pared trata de moverse hacia dicha pared, el movimiento lógicamente no se realiza, pero tampoco se encara a dicha pared. Esto se soluciona alterando el orden de las acciones, primero el personaje se encara y después se ejecuta el movimiento. Anteriormente, si el movimiento no se podía realizar, tampoco se encaraba.

- Aunque se cambie el componente Look de Game, la cámara sigue enfocada en el primer objetivo.

La cámara mantenía los atributos iniciales, aunque alguno de ellos se cambiase. Se desarrolló un módulo que desactivaba la cámara y generaba una nueva y estuvo funcionando hasta que los desarrolladores arreglaron el problema.

- La primera vez que en el juego se selecciona una celda inaccesible, el juego crashea durante varios segundos.

Este error fue comunicado a los desarrolladores y solucionado por ellos. Se basaba en que el juego buscaba en todas las celdas creadas y trataba de aplicar en ellas el algoritmo A* para encontrar un camino que permitiese moverse hasta ella.

A continuación se van a comentar algunos errores que aún no han sido solucionados:

- Si se cambia de escena y quiere volverse a una escena anterior esta aparece vacía.

Esto es debido a que *IsoUnity* guarda el estado actual de las escenas y que no permite reiniciarlas. Es un grave error ya que obliga a que cada vez que perdamos la partida tengamos que cerrar el juego y volver a abrirlo.

- En mapas grandes aparecen líneas sin pintar en ciertos lugares. Dichos espacios aparecen perfectamente pintados si se utiliza la misma textura pero en un mapa más pequeño.

No se sabe a qué es debido dicho error. No es un gran problema, pero afecta estéticamente al juego.

6.3.- Problemas y requisitos adicionales encontrados

Como ya se ha dicho anteriormente se planteó un sistema de entregas evolutivas. La idea era subir una primera versión del juego y recibir las críticas en forma de elementos a cambiar, añadir o eliminar y con ello ir progresando. El objetivo es una versión final con la mayor cantidad de iteraciones, lo que produciría una gran mejora. A continuación pueden verse distintas mejoras planteadas a lo largo de las iteraciones:

- Adso no se mueve una vez que se llega a la habitación de Guillermo

En el día 1, el primer objetivo es ser guiado por el Abad hasta llegar a la celda destinada a Guillermo. Una vez allí, se podía seleccionar a un personaje u otro, pero Adso no se podía mover. Esto originó el descubrimiento de uno de los errores comentados en el punto 6.2. Este será el único error de este tipo que se comentará ya que provocó un enorme retraso en el avance del proyecto debido a que fue necesario revisar gran parte del código para ver dónde podía estar el problema, hasta descubrir que era un error en *IsoUnity*.

- Añadir un modo exploración que permita visitar el mapa de la abadía, su objetivo es comprobar “inGame” el estado del mapa de la abadía.

No es útil para el usuario final, aunque se ha determinado una posible visita a la abadía sin interacción con los personajes.

- El abad se mueve demasiado rápido y no da tiempo al jugador de seguirle.

Se soluciona añadiendo nuevos estados de control y parada para facilitar que un jugador inexperto pueda seguir al abad.

- El abad tarda mucho en reaccionar cuando se llega hasta él.

La solución a este problema pasa por añadir una nueva condición a esos estados que controlan la distancia entre el abad y el jugador. Una vez que se alcanza la distancia deseada aunque no se haya alcanzado el tiempo de espera relativo al estado, se puede avanzar de estado.

- Distintas modificaciones en la música en la búsqueda de una más adecuada.

- Verificar que las imágenes o no tienen copyright o son del juego original.

Se modificaron varias imágenes para no incurrir en problemas futuros.

- Errores relativos al control de los distintos personajes.

- Corregir colisiones de varias decoraciones.

Esto se soluciona desmarcando la casilla “walkable” de diversas celdas relativas a este problema.

- Corregir diversos problemas del mapa por donde los protagonistas pueden subirse y no deberían.

- Hacer menos estricto el nivel de seguimiento del abad.

En el GitHub del proyecto pueden encontrarse más tareas en las que se está trabajando actualmente, ya que este no es un proyecto que quiera abandonarse una vez entregado el TFG, sino que se espera pueda llegar a terminarse en algún momento. De esta forma es un proyecto en continuo crecimiento en el que nuevo FeedBack es incorporado cada poco tiempo y nuevos elementos están siendo desarrollados.

6.4.- *IsoUnity* en un entorno de producción

A continuación se va a proceder a la evaluación de *IsoUnity* desde distintas perspectivas, se espera que dicho informe sirva a alguna empresa que quiera utilizar la herramienta *IsoUnity* en un entorno de producción.

En cuanto a las opciones de mercado *IsoUnity* se encuentra limitado a hispanoparlantes, ya que no se encuentra traducido a ningún otro idioma (tampoco lo está el manual). Esto disminuye la visibilidad, por la tanto debería traducirse a otros idiomas.

El diseño de mapas carece de cualquier tipo de realimentación. Sólo se puede aprender a generar un mapa haciendo varios, no se dispone de ningún tipo de material ilustrativo en el que se explique el proceso. En el anexo correspondiente al manual se incluye una documentación básica, por lo que sería necesario incluir algún tipo de vídeo en el que se explique cómo hacer un mapa con las funciones básicas de éste.

Todavía tiene que mejorarse el rendimiento de *IsoUnity*, ya que existen caídas inesperadas de frames de las que aún se desconoce el motivo. Esto puedo limitar mucho la elección de este motor por parte de una gran empresa.

El sistema de creación y gestión de sprites ha de ser mejorado. La creación de los mismos es engorrosa (Se puede tardar unos 7-10 minutos en hacer un único sprite con una imagen ya hecha) y puede llegar a alargar mucho el diseño.

Los eventos tienen que ser reacondicionados para incluir numerosas circunstancias que no se acometen actualmente, desde la inclusión de poder encadenar varias hasta su GUI, la cual presenta errores.

Algunas funciones básicas como el movimiento deberían estar preimplementadas y utilizarse con un solo botón en el que seleccionar los distintos atributos. Siguiendo esta temática deberían incluirse distintos ejemplos de cómo realizar dichas acciones.

Pero no son todos aspectos negativos, el GitHub de la herramienta se encuentra muy activo y los errores suelen resolverse de forma muy rápida. La herramienta cuenta con una indudable proyección y aunque se encuentra actualmente en una fase inicial cabe considerar que a día de hoy es el único plugin sobre Unity en su estilo.

Definitivamente consideramos que después de pasar un año trabajando con ella, desde sus momentos más duros, que es una herramienta que puede ser utilizada por cualquier usuario, incluso una empresa dedicada al ámbito de los videojuegos. Únicamente necesita un poco más de mimo y mejora para convertirse en la enorme herramienta que está destinada a ser, ya que potencial no le falta.

Capítulo 7. Conclusiones

Este es el último capítulo del TFG, aquí se desarrollaran las conclusiones de este proyecto así como la mención de las tareas en las que ha participado cada uno de los integrantes.

7.1.- Conclusiones

El proyecto desarrollado ha tenido su motivación en un interés por los videojuegos clásicos y sus reconstrucciones mediante tecnologías actuales. Básicamente, ha consistido en utilizar una herramienta llamada *IsoUnity* creada por unos alumnos de la Facultad como parte de su Trabajo Fin de Grado del curso 2013-2014, cuyo objetivo era explorar la creación de videojuegos isométricos con estética retro. Nuestro trabajo ha tratado de someter dicha herramienta a un proceso lo más exigente posible y desarrollar con él el prototipo de lo que sería una reconstrucción del conocido título de los años 80, *La Abadía del Crimen*. Para hacer dicha reconstrucción se ha realizado un detallado estudio sobre el mencionado videojuego, el cual podrá ser utilizado por cualquier estudio profesional de videojuegos. El estudio cuenta con todas las características reseñables del juego original así como todos los cambios que deberían aplicarse para hacerlo viable para el público actual.

Este proyecto ha seguido el ciclo de vida típico de cualquier aplicación que pueda ser desarrollada a nivel profesional, intentando poner en práctica algunas técnicas de Ingeniería del Software para su gestión y documentación, principalmente metodologías ágiles. Esto es interesante ya que facilita su seguimiento y replicación.

El trabajo que se realizó sobre *IsoUnity* demandó nuevas características que han sido aplicadas y aprovechadas a la hora de desarrollar nuestro prototipo, favoreciendo que durante este año la herramienta haya evolucionado considerablemente. Esto puede verse en los nuevos módulos y elementos desarrollados así como en las diversas versiones por las que ha pasado *IsoUnity* desde septiembre de 2014, cuando comenzó el proyecto.

Por otra parte es interesante mencionar que las ideas anteriormente desarrolladas pueden servir como base para ser utilizadas por cualquier compañía que decida embarcarse en el proyecto de hacer la reconstrucción, ya que utilizándolas consideramos que a partir de ellas acabar la reconstrucción es solo cuestión de tiempo. No hay que desarrollar elementos nuevos, se pueden utilizar los elementos existentes e ir montando la aventura.

El estudio de *La Abadía del Crimen* se encuentra dividido en dos partes: En el capítulo 4 se encuentran los elementos más importantes del juego original y en el Anexo 3 se puede encontrar una guía completa sobre la actividad de los monjes catalogada por días y horas. Consideramos que se deben tener en cuenta las sugerencias aportadas sobre los cambios introducidos, ya que además de seguir modelos que se pueden encontrar en otros videojuegos actuales tratan de mantener la esencia del juego original para despertar la nostalgia de los jugadores.

Entre los elementos desarrollados cabe recalcar la importancia de las distintas funciones que generan movimiento en los personajes, así como el sistema de inventario y de conversación (chat) paralelos a los que ofrecía *IsoUnity*. Las optimizaciones en *IsoUnity* han convertido la herramienta en un motor plenamente funcional y aunque todavía no es una herramienta que haya sido utilizada en entornos profesionales, podemos decir que va por un excelente camino.

En cuanto al estado de *IsoAbbey* (la reconstrucción) se puede decir que es algo más que una demo ya que permite jugar el Primer Día del juego de forma completa. Se han desarrollado todas las funciones principales que tiene el juego original, aunque no han podido ser implementadas en el propio videojuego, por tanto se suministra con una pequeña aplicación adicional. Dicha aplicación muestra un entorno austero en el que se ilustran con ejemplos dichas funcionalidades principales de *La Abadía del Crimen*.

El Anexo 4 contiene un completo manual sobre *IsoUnity* que incluye todo lo necesario para comenzar a desarrollar cualquier tipo de aplicación basado en dicha herramienta. Adjunta imágenes que ayuda en todo momento a su mayor comprensión.

En resumen consideramos que *IsoUnity* es ahora mismo una herramienta válida para la creación de juegos en perspectiva isométrica. También creemos que el estudio sobre *La Abadía del Crimen* original que hemos realizado en este proyecto puede servir a una empresa para llevar a cabo la reconstrucción.

Para terminar nos gustaría dar nuestro agradecimiento a Risin´Goat, una empresa indie de videojuegos, que ha valorado muy positivamente éste proyecto, como puede verse en el Anexo 3, el cual contiene una carta escrita por su fundador. En dicha carta se destaca la viabilidad de este proyecto como punto de partida para cualquier desarrollador que desee embarcarse en un proyecto similar.

7.2.- Aportaciones de los integrantes

A continuación se enumeran las tareas realizadas por cada integrante del TFG en el desarrollo del mismo.

7.1.1.- Antonio Santamaría Barcina

Elementos que no aparecen en el TFG actual

El proyecto actualmente desarrollado no es similar a lo que se iba a hacer en un primer momento. Una gran parte del trabajo realizado en los primeros meses no aparece reflejado hoy en día debido a que fueron ideas desechadas debido a cambios de decisión o porque su implementación fue imposible debido a motivos técnicos.

En un primer momento el videojuego que se iba a desarrollar iba a ser algo parecido a la búsqueda del ladrón. El jugador iba a controlar al bibliotecario e iba a tratar de descubrir al asesino, el cual se suponía que iba a cambiar de partida en partida. El juego iba a mantener una serie de hechos comunes (de asesinatos) entre los que se elegiría en cada

partida. Me encargué de desarrollar los árboles que mostrasen todas las posibilidades para diferentes elecciones de asesinatos.

El bibliotecario era ciego, por lo que me propuse a representar esto. Planteé la idea de un animal que acompañase al bibliotecario y le avisase de cuando estaba cerca de algún otro monje, éste no iba a saber cuál de los monjes era y a través de las distintas pistas que iba encontrando tendría que descubrir quién era el asesino. Este modelo se acabó desechando debido a que se pensaba integrar un prototipo de sistema de Inteligencia Artificial, pero que por diversas causas técnicas no se pudo culminar.

Una vez que se fijó el prototipo actual me encargué crear los mapas del juego. Esto pasó por varias fases debido a que había que elegir entre diversos modelos y testearlos para encontrar limitaciones de *IsoUnity.* Diseñé mapas con celdas separadas como teletransportadores, mapas con exteriores y el diseño actual que hemos adoptado.

Elementos que aparecen en el TFG actual

Mi trabajo fue el de desarrollar el mapa que utilizaríamos para hacer *IsoAbbey* (creación, pintado y decoración), el mapa no contiene únicamente los elementos visuales que se pueden ver a simple vista sino también una gran cantidad de componentes de *Unity* que ayudan a gestionar las distintas secuencias que se pueden ver en el Día 1.

En cuanto a la parte de la construcción de *IsoAbbey* me encargué de escribir el código necesario en C# que hace posible que este funcione. Comencé basándome en los elementos de *IsoUnity* y posteriormente fui implementando las ideas que iban apareciendo en las reuniones gracias a las aportaciones de los miembros.

Después de tanta interacción con IsoUnity comencé a encontrar varios errores, ayude a los desarrolladores a localizar dichos problemas y con la solución de algunos de ellos, así como diversas optimizaciones del código original. Por otra parte rediseñé algunas de las texturas que quedaron inservibles después de ciertas actualizaciones realizadas sobre *IsoUnity*.

También me encargué de desarrollar el estudio del día 1 para poder realizar su implementación en *IsoAbbey*. Esto lo hice a partir del visionado de diversos GamePlays y jugando al propio juego. Fue así como obtuve la mayor parte de las ideas que se cambiaron en este día, ya que cogí todo aquello que no me gustaba y traté de cambiarlo de forma que me gustase más jugarlo.

Con respecto a la memoria he escrito los capítulos 5 y 6 y ayudé a Alex a revisar los puntos 3 y 4. Y por último me encargué de la maquetación y corrección de la memoria (Capítulos y Anexos).

7.1.2.- Alejandro Alexiades Estarriol

Aporté diversas ideas en la elaboración de los primeros prototipos del TFG que no se encuentran presentes en la actualidad.

Me encargué de la documentación y memoria. En cuanto a la memoria realicé los capítulos 1, 2, 3 y 4. Por otra parte realicé ambos anexos (análisis de los días en *La Abadía del Crimen* y el manual para el uso e instalación de *IsoUnity*) así como la Bibliografía.
También me encargué de la obtención de texturas, personajes y decorados que se pueden ver en *IsoAbbey*.

Bibliografía

Crytek (2008): Transition to scrum midway through a aaa development cycle: Lessons learned. In Game Developer Conference, Marzo 2008. [Accessed: 21-Aug-2015].

Demachy, T. (2008), Extreme game development
Disponible en: http://www.gamasutra.com/resourceguide/20030714/demachy01.shtml. [Accessed: 21-Aug-2015].

DEV (2015): Asociación Española de Empresas Productoras y Desarrolladoras de Videojuegos y Software de Entretenimiento
Disponible en: http://www.dev.org.es/es/publicaciones/libro-blanco-dev. [Accessed: 17-Jul-2015]

Eclipse Foundation (2008): Eclipse process framework project homepage
Disponible en: www.eclipse.org/epf/. [Accessed: 21-Aug-2015].

Esteve, J. (2014) Obsequium: Siete puntos de vista diferentes de La Abadía del Crimen desde el contexto histórico al racionalismo tecnológico pasando por los arrebatos sentimentales que provoca el juego de Francisco Menéndez y Juan Delcán.
Disponible en: http://www.ochoquilates.com/products/obsequium. [Accessed: 6-Jul-2015].

Esteve, J. (2012) Ocho Quilates: Repaso, con infinidad de testimonios de la época, a una de las etapas más brillantes de producción de videojuegos en España.
Disponible en: http://www.ochoquilates.com/collections/productos/products/ocho-quilates-vol-1-ebook [Accessed: 6-Jul-2015].

FIFE Team (2013), Flexible Isometric Free Engine - game engine
Disponible en: Available: http://www.fifengine.net/ [Accessed: 21-Aug-2015].

Filmation (1984): Filmation Engine - game engine
Disponible en: Available: http://pages.rediff.com/filmation-engine/550763 [Accessed: 05-Jul-2015].

Foe,S. (2008), Gamasutra. Interview: Nokia's - a member of the reset generation.
Disponible en: http://www.gamasutra.com/php-bin/news index.php?story=19210. [Accessed: 21-Aug-2015].

Galvany,R.(2006), Desarrollado del remake del knight Lore
Disponible en: *http://ima.udg.edu/~dagush/Projects/KnightLore2006/* [Accessed: 15-Jun-2015].

Irrelon Software, (2013), Isogenic Game Engine - game engine
Disponible en: Available: http://www.isogenicengine.com/ [Accessed: 21-Aug-2015]

Keith, C. (2008), an agile restrospective. In Game Developer Conference [Accessed: 21-Aug-2015].

Keith, C. (2009), advanced scrum and agile development. In Game Developer Conference [Accessed: 19-Aug-2015].

Newzoo (2015), Consultora tecnológica.
Disponible en:. http://www.newzoo.com/. [Accessed: 17-Jul-2015]

Nimrod (2008): Nimrod - game development tool
Disponible en: Available: http://herr-o.deviantart.com/art/Nimrod-the-isometric-editor-88131824 [Accessed: 21-Aug-2015].

Nutt, C. (2008), Living on the edge: Dice's owen o'brien speaks
Disponible en: http://www.gamasutra.com/view/feature/3684/living. [Accessed: 21-Aug-2015].

Nvidia (1993): Empresa multinacional especializada en el desarrollo de unidades de procesamiento gráfico y tecnologías de circuitos integrados
Disponible en: Available: http://www.nvidia.es/page/home.html4 [Accessed: 21-Aug-2015].

Object Managment Group (2007): Software and systems process engineering metamodel specification, version 2.0. [Accessed: 21-Aug-2015].

Pérez Colado, I.J., Pérez Colado, V.M. (2014): Un conjunto de herramientas para Unity orientado al desarrollo de videojuegos de acción-aventura y estilo retro con gráficos isométricos 3D. Trabajo Fin de Grado en Ingeniería del Software, Facultad de Informática, Universidad Complutense de Madrid. E-Prints UCM, Madrid, España
Disponible en: http://eprints.ucm.es/30160/ [Último acceso: 05-Jul-2015].

Serrano Acosta, F.A (2014), El Arte de las Portadas
Disponible en: http://www.meristation.com/pc/reportaje/el-arte-de-las-portadas/1975476/58

Schwaber, K. and Beedle, M. (2001), Agile Software Development with Scrum. Prentice Hall PTR. [Accessed: 21-Aug-2015].

Sinclair Research, (1982): Sinclair ZX Spectrum - Personal computer
Disponible en: Available: http://www.worldofspectrum.org/ [Accessed: 05-Jul-2015].

Tobey, B. (2008): Introducing scrum at large animal games: a look back at the first year of agile development
Disponible en: http://www.gamasutra.com/view/feature/3677/introducing. [Accessed: 21-Aug-2015].

Anexos

Anexo 1.- Title (in English)

Reconstruction of a classic isometric video-adventure using *IsoUnity* as a production tool platform

Anexo 2.- Conclusions (in English)

The project has been developed motivated by an interest in classic video games and reconstructions using current technologies. Basically, it has been to use a tool called IsoUnity created by some students of the Faculty as part of their Final Project of the year 2013-2014, which aimed to explote the creation of isometric game with retro aesthetics. Our work has shought to bring the tool to process as demanding as possible and develop with it the prototype of what would be a reconstruction of the famous title of the 80s, *La Abadía del crimen*. For doing this reconstruction we have carried out a detailed study about the metioned game, which it may be used by any profesional game studio. The studio has all the notable features of the original game as well as all the changes that should be implemented to make it viable for today's audiences.

This project has followed the typical life cycle of any application that can be developed professionally, trying to implement some software engineering techniques for management and documentation, mainly agile software development. This is interesting because it facilitates monitoring and replication.

The work performed on IsoUnity demanded new features that have been implemented and exploited when developing our prototype, favoring considerably the evolvement of the tool during this year. This can be seen in the new developed modules and components, as well as in the various versions bore by IsoUnity since September 2014, when the project began.

Moreover it is interesting to note that the ideas developed above can serve as a base to be used by any company which decides to embark on the project of making the reconstruction, because using them is only a matter of time. It is not necessary to develop new elements, these are necessary for creating the adventure

Among developed elements it should be emphasized the different functions that generate movement in the characters and the inventory and conversation system (chat) parallel to those offered by IsoUnity. IsoUnity optimizations have turned the tool into a fully functional engine that can be used for creation of professional games, though not being tool used in professional environments, we can say that it is on a excellent path.

The study about La Abadía del Crimen is divided into two parts: In chapter 4 are the most important elements of the original game and in Annex 3 we can find a complete guide about the activity of the monks classified by days and hours. We believe that both should be taken into account the suggestions made on changes, as well as role models to be found in other existing games try to keep the essence of the original game to awake the nostalgia of the players.

Among developed elements it should be emphasized the different functions that generate movement in the characters and the inventory and conversation system (chat) parallel to those offered by IsoUnity. IsoUnity optimizations have turned the tool into a fully functional engine that can be used for creation of professional games, though not being tool used in professional environments, we can say that it is on a excellent path.

Regarding the status of reconstruction, which we called *IsoAbbey*, you can say it's more than just a demo as it allows to play the first day completely. All the main functions of the original game have been developed, although couldn´t be implemented in the game itself, so it's supplied with a small additional application. This additional application displays a sober environment in which examples illustrate the main features of La Abadía del Crimen.

Annex 4 contains a comprehensive manual about *IsoUnity* that includes everything needed to start developing any application based on such a tool. Attached images that help at all times to their greater understanding.

As summary we believe that IsoUnity is now a valid tool for creating isometric games. We also believe that the study we have on the original done on the original Abbey of Crime in this project can be useful for a company to perform the reconstruction.

Finally we would like to thank Risin'Goat, an indie video game company, which has very much appreciated this project, as shown in Annex 3, which contains a letter written by its founder. In that letter the viability of this project as a starting point for any developer wishing to embark on a similar project stands.

Anexo 3.- Carta de Risin´Goat

En ocasiones es difícil para una empresa indie dedicar todo el tiempo necesario a la investigación de un proyecto. Hay que tener en cuenta que, sobre todo al principio, es complicado cuadrar las inquietudes creativas de del equipo con la necesidad de supervivencia por parte del estudio como entidad profesional, el cual necesita unos ingresos mínimos y medianamente constantes.

En este punto, la universidad puede jugar un papel muy importante tendiendo un puente entre ambas realidades, dado que dispone de más tiempo para dedicarle a este tipo de procesos. No sólo eso, la investigación forma parte del ADN de la universidad, por lo que en determinados aspectos, los recursos y el enfoque universitarios son más adecuados para este tipo de tareas.

Por supuesto, creo firmemente que las ventajas son recíprocas, pues una colaboración de estas características permite a los alumnos asomarse a la realidad de los estudios indies de videojuegos. Realidad que muchos están deseando experimentar. No sólo eso, podría ser la base para entrar directamente en uno de ellos nada más acabar la carrera, incluso de llevar sus proyectos más allá con apoyo profesional. En cualquier caso, crear contactos desde antes de acabar la formación universitaria siempre es una ayuda.

En el caso particular que nos ocupa, La Abadía del Crimen, una investigación pormenorizada tiene el valor adicional de hacer hincapié en los videojuegos clásicos y sus remakes. Desde luego los puntos de vista son interesantes: entender los orígenes de la industria en nuestro país, conocer los problemas que tenían entonces los desarrolladores, ayudar a mantener viva la historia de los videojuegos, definir la dificultad que puede entrañar hacer un remake que sea jugable bajo los estándares de calidad actuales...

Todos estos aspectos y otros se contemplan en esta memoria, convirtiéndola en un buen punto de partida para un desarrollador interesado en embarcarse en un proyecto similar.

Escrito por: Carlos L. Hernando - Fundador, Propietario y Diseñador de Videojuegos en Risin'Goat Indie Game.

Anexo 4.- Manual de *IsoUnity*

IsoUnity para mortales

Guía para el manejo sencillo e intuitivo del paquete *isoUnity* para *Unity*

Índice:

1. **Que es *IsoUnity***
2. **Instalación de *IsoUnity***
3. **Creación de mapa**
4. **Creación de celda**
5. **Aplicación de texturas a celdas**
6. **Aplicación de decorados a los mapas**
7. **Entidades**
8. **La clase gestora Game**
9. **Implementando eventos en el juego**
10. **Implementando las entidades**
11. **Los componente jugador, player**
12. **La componente movedora, mover**
13. **La componente habladora, Talker**
14. **La componente almacén, Inventory**
15. **La componente objeto, ItemScript**
16. **La componente RandomMove**
17. **La componente Teleporter**
18. **Manejadores de eventos**
19. **Implementación del controlador**
20. **Implementación de interfaces**
21. **Interfaz de diálogos**
22. **Interfaz de inventario**
23. **Interfaz de selección de acciones**
24. **Interfaz de controles de pantalla**
25. **Implementación de secuencias**

26. **Interprete de secuencias**
27. **Editor de secuencias**
28. **Editor del nodo dialogo**
29. **Editor del nodo eventos**
30. **Editor de bifurcación**

1. - Que es IsoUnity

Un paquete isométrico de estilo retro para Unity. Creado por Ivan Jose Perez Colado y Victor Mauel Perez Colado.

Se pueden crear videojuegos utilizando esta herramienta, sin necesidad de tener conocimientos de programación. Únicamente, el usuario deberá aprender a utilizar a grandes rasgos el motor Unity y esto se puede aprender a utilizar gracias a este proyecto.

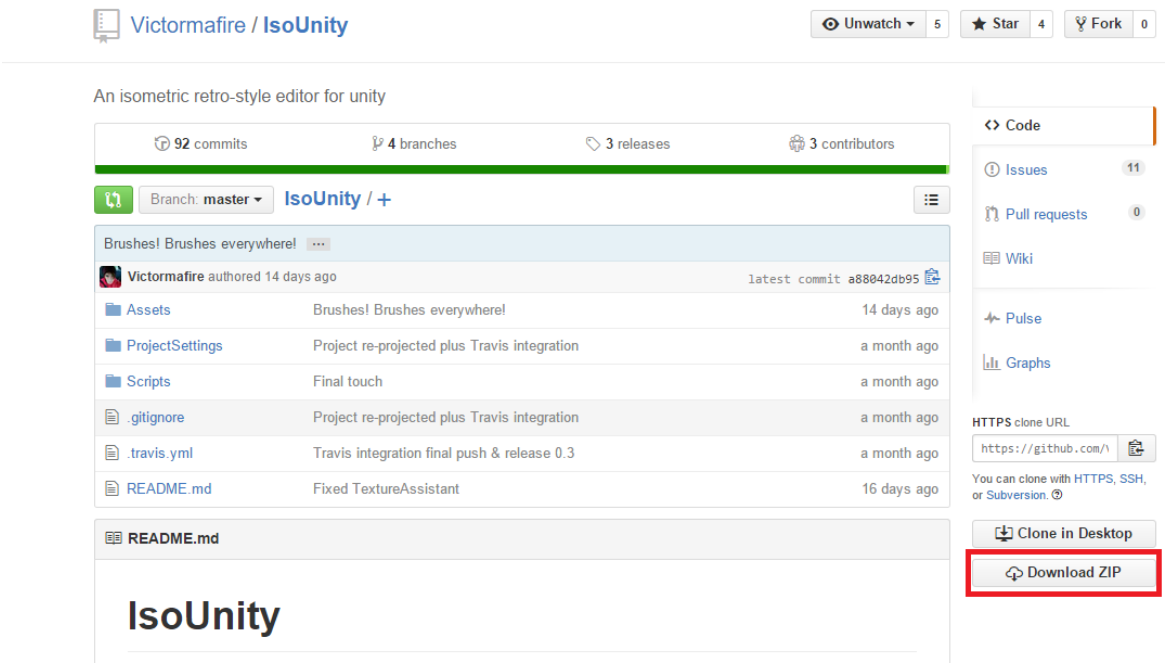
Los videojuegos realizados con esta herramienta pueden hacerse implementando diversos hitos. El progreso en la creación de ese juego puede medirse gracias a la completitud de dichos hitos. También permite desarrollar historias interactivas en las que el jugador tomará parte.

2. - Instalación de IsoUnity

Una vez instalado *Unity* hay que aplicar el paquete de *IsoUnity*, que puede conseguirse de las siguientes formas:

1. **Internet**.
   a. Descargar IsoUnity de la url: https://github.com/search?utf8=%E2%9C%93&q=isounity
   b. Desde el programa Unity hacemos:
      File-> Open Project -> Open Other (botón) -> Buscamos la carpeta IsoUnity->Presionar botón Seleccionar carpeta.
   c. Se reiniciara *Unity* cargándose el paquete IsoUnity

2. **Paquete Unity**. (Recomendado)

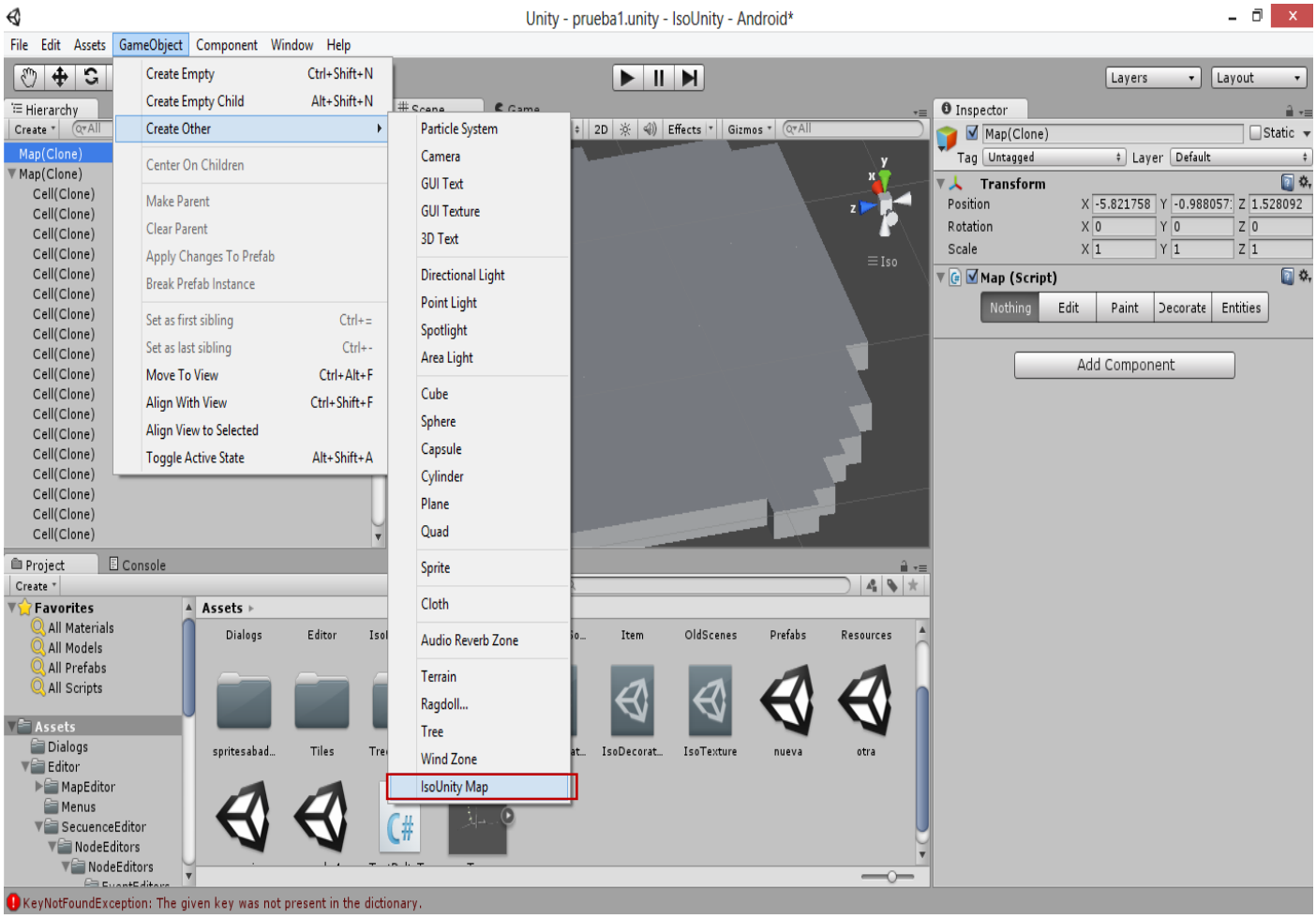


Página web de Github donde se descarga de IsoUnity

3.- Creación de Mapas

Para poder crear un mapa primeramente se tiene que crear el elemento *IsoUnityMap*, el cual puede encontrarse dentro de *GameObject* y a continuación en *Other*. Este es un *GameObject* sobre el que se van añadiendo las Cell que conforman el mapa.

Diferentes mapas pueden coexistir a la vez en una misma escena, pero la cámara solo puede enfocarse en uno de los mapas, el cual coincide con el del Jugador. Un posible problema es que los mapas no enfocados se desactivan, por lo que los NPC no pueden moverse ni realizar ninguna acción por ellos.



4.- Creación de Celdas

Una celda es el componente básico de *IsoUnity*. Tiene una forma cuadrangular, de una celda suelta pueden verse 3 caras, las cuales pueden pintarse.

Cada celda permite modificar su altura y el acabado, es decir, plano inclinado o máxima inclinación. También posee un componente denominado “walkable” el cual indica si un personaje puede andar o no sobre esa celda.

Para crear una celda se puede hacer de la siguiente forma: Primeramente hay que seleccionar el mapa **IsoUnityMap.** En la ventana *Hierachy* aparecerá *Map(clone).*

1. Seleccionar Map(clone)
2. En la ventana Inspector pulsar en el botón *Edit* que está en *Map(Scrip*)
3. Aquí se podrá editar la altura de la celda en *Grid Height*. Por defecto está a 0.
4. Luego en la ventana Scene con el ratón y presionando el botón izquierdo se irán añadiendo celdas.

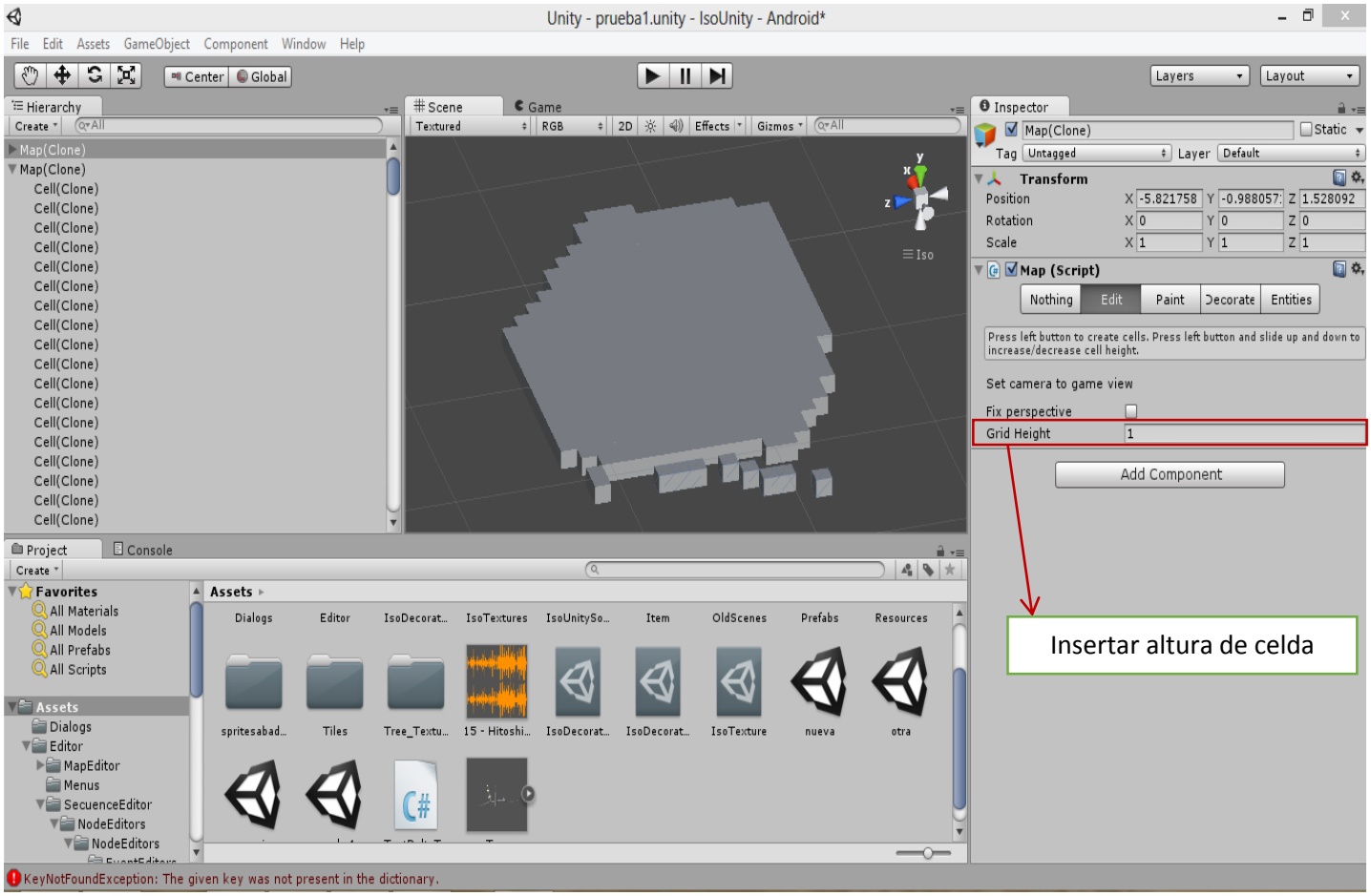
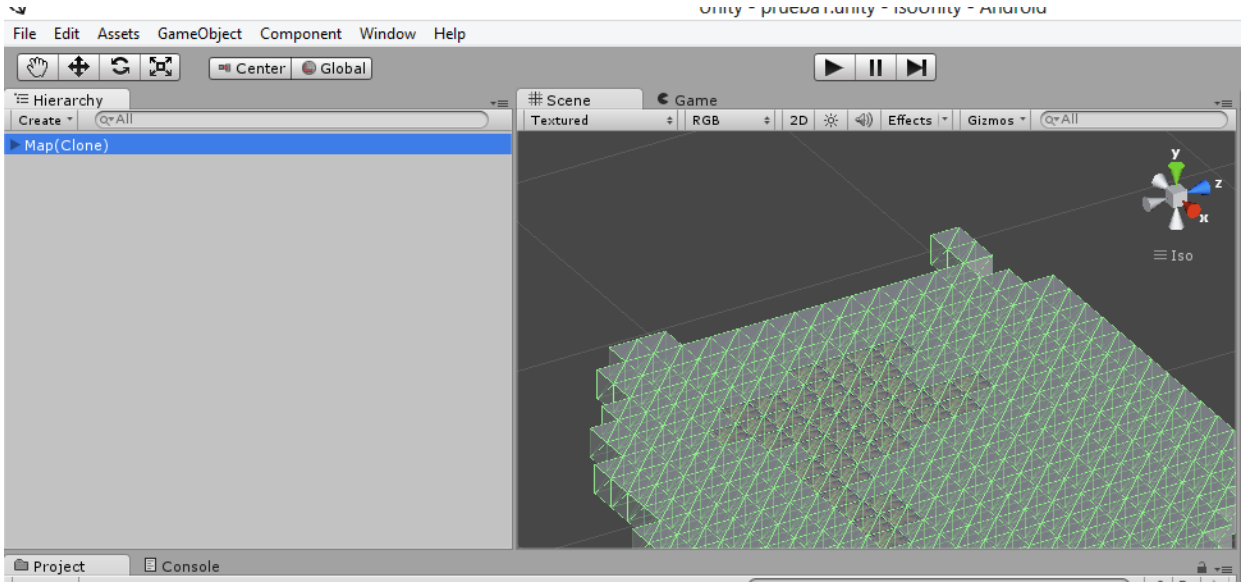


Imagen que muestra la creación de celdas por medio de IsoUnityMap

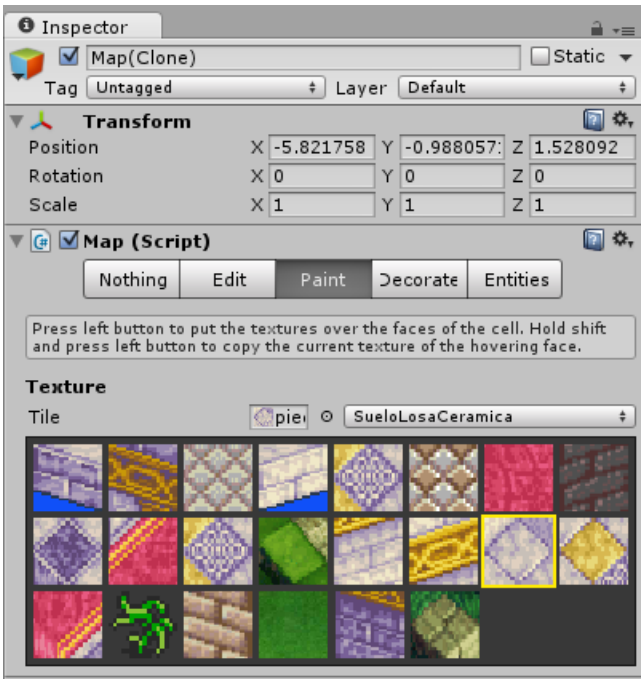
5.- Coloreado de Celdas

El pintado de celdas puede hacerse mediante la ventana de edición de texturas, la cual se encuentra en una pestaña junto a la creación de la celda, sus características. El proceso es el siguiente:

- Seleccionar en la pestaña *Hierarchy Map (clone)* donde seguidamente todas las celdas creadas se pondrán de color verde para poder seleccionar las caras.



- En la ventana Inspector dentro de Map(Script) seleccionar el botón Paint
- Aparecerá una galería de imágenes sobre las que pulsar:



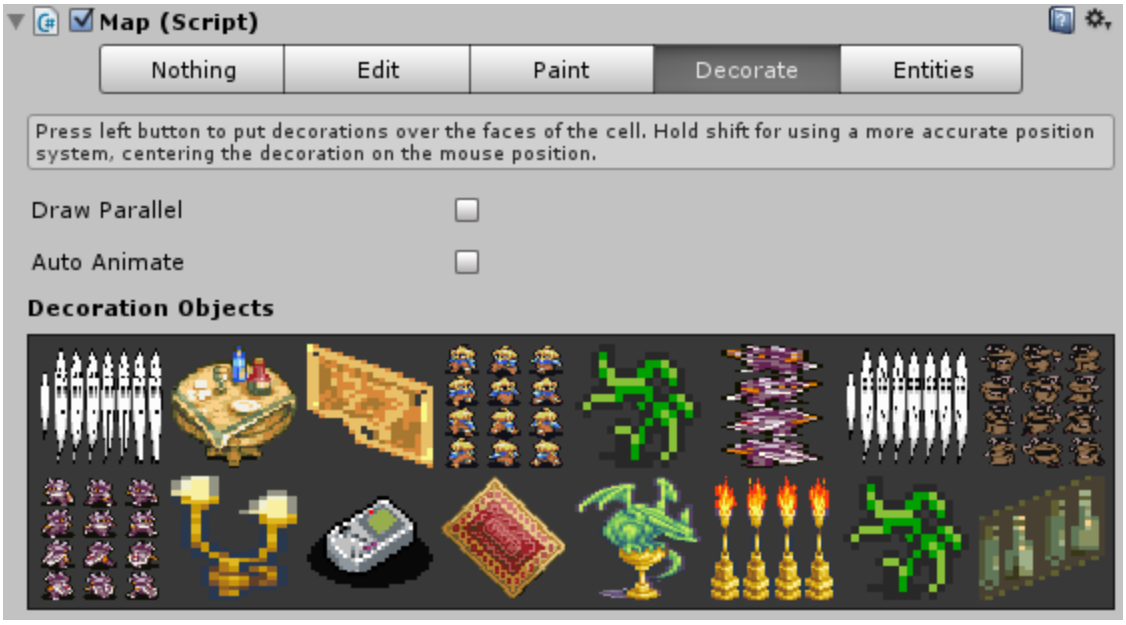
- Después de seleccionar una de las texturas se puede elegir si se desea pintar una única celda o toda una superficie. Tras esto solo hay que pulsar sobre las celdas para aplicarles una textura.

6.- Aplicación de Decorados a los mapas

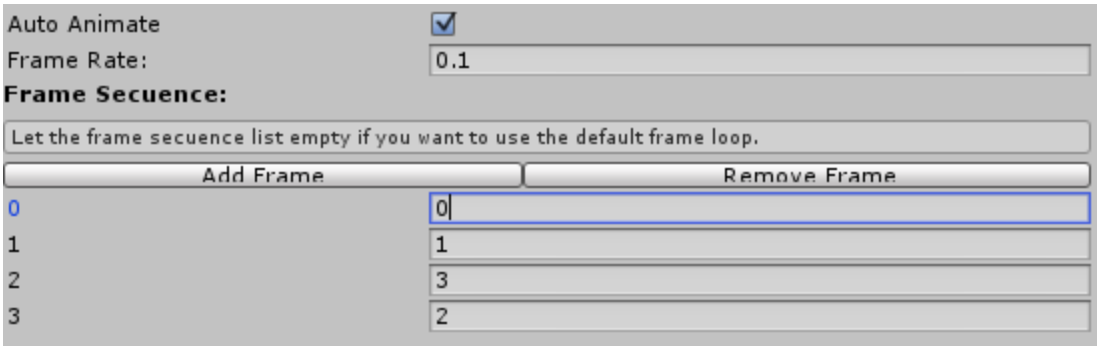
El funcionamiento del decorado es similar al de pintado. El proceso a seguir para colocar un decorado en el mapa es muy similar al de pintar una casilla, se hace a partir de una pestaña similar a la del pintado.

En primer lugar, tras un breve texto que explica el funcionamiento del módulo, se muestran dos opciones que permiten establecer: Si la decoración se va a colocar de forma paralela a la cara o no, y si se desea que la decoración sea animada o no.

En la parte inferior de estas opciones, se muestra un selector visual similar al selector de IsoTexturas que se encuentra en el módulo de pintado del mapa, con la diferencia de que este selector, en lugar de mostrar la texura de una IsoTextura, muestra la textura de una IsoDecoración. El resultado de este inspector puede verse a continuación:



La segunda de las opciones, la opción que permite establecer si una decoración está animada, al activarse, muestra una nueva sección en la interfaz. En dicha sección se permite personalizar el comportamiento de dicha animación estableciendo: El tiempo de duración de cada fotograma en segundos, en Inglés *frame rate*, y una lista de fotogramas, en el orden en el que se deben producir para realizar la animación. Esta lista, si se mantiene vacía, producirá una secuencia de fotogramas por defecto, comenzando por el primero, y, al terminar, volviendo a éste. Este panel puede verse a continuación:



Este apartado de animado, hace que, en el momento de instanciar una decoración, se instancie con un componente adicional *AutoAnimator* que se encarga de realizar dicha animación.

Una vez se encuentre parametrizada y seleccionada la decoración que se desea colocar en el mapa, al colocar el cursor sobre el mismo, se muestra la Decoración Fantasma. Esta Decoración Fantasma se diseñó en el capítulo 4, y se ha implementado su funcionalidad por completo, estableciendo un comportamiento similar al de la Celda Fantasma. Esta decoración es especial, pues su representación es parcialmente transparente, y se elimina automáticamente cuando se abandona este editor, o se inicia el juego. La función de esta decoración es la de mostrar, antes de instanciar dicha decoración en el mapa, una representación del resultado que se va a obtener, y por consiguiente, facilitar la tarea de decorado al usuario.

Como mecánicas adicionales, los modos de posicionamiento de decoraciones: centrado en la celda, o centrado en la posición del cursor, son accesibles pulsando o no la tecla Mayús., en Inglés *Shift*, del teclado. Cuando dicha tecla no está pulsada, se produce un posicionamiento centrado en la cara de la celda donde se va a posicionar, y, cuando sí está pulsada, este posicionamiento se realiza centrando en la posición del cursor.

Finalmente, para remarcar exactamente la cara en la que se va a posicionar la decoración, se dibuja un borde blanco en la misma. Esta característica se muestra en cualquiera de los modos de posicionamiento de decoración.

7.- Entidades

El módulo de entidades es el más simple de todos. Su funcionalidad no dispone de creación de entidades, pues es complejo de implementar y Unity ya provee herramientas suficientes para llevarlo a cabo.

Dentro de la parte de inspector, para realizar su funcionamiento simplemente contará con un campo en el que se podrá seleccionar una entidad, siendo ésta la que se posiciona.

Dentro de la parte de escena, de una forma similar a como se hace en el módulo de pintado, se selecciona la cara superior de la y tras situarse en ella se podrá colocar la entidad encima.

De esta forma, facilitamos la capacidad para disponer las entidades en el mapa haciéndolo levemente más simple.

Para crear las entidades, el método más simple consiste en la creación de una decoración con la apariencia que se desea dar a la entidad, en la celda que se desee, seleccionarla con el cursor y añadirle la componente *Entity*.

8.- La clase gestora Game

Esta clase encarga de mantener el bucle principal de la lógica del juego. *Unity* ya proporciona un bucle principal, sin embargo, la necesidad de *Game* radica en controlar dicho desarrollo, permitiendo la distribución de actualizaciones y eventos a las distintas partes que conforman el proyecto.

En primer lugar, *Game* será el encargado de configurar la cámara en su perspectiva, de ocultar y desactivar los mapas marcados como tal y de centrar la cámara en el jugador. Por último, creará los gestores de eventos que se necesiten. Al finalizar, comenzará el bucle principal que pasará por las siguientes partes:

1. **Actualización del controlador**: Dado que la entrada de datos por parte de una persona está centralizada en el controlador, se le manda a éste que realice su actualización. Durante ésta, el controlador recolectará datos de entrada y los distribuirá entre los distintos oyentes. En próximos capítulos se detalla más el funcionamiento.
2. **Distribución de eventos a los manejadores**: Dado que las partes del proyecto se comunican a través de Eventos, se distribuirán los eventos generados desde el último *tick* a los distintos manejadores. Algunos manejadores son el de animaciones, el de secuencias y el de mapas.
3. **Notificación de actualización a los distintos gestores**: Tal y como en la distribución de eventos, iterativamente se avisa a todos los gestores que se está produciendo una actualización.

Dentro del bucle principal de Unity, al finalizar la actualización de Game descrita en el bucle anterior, se ejecutará varias veces la actualización de la GUI. Por ello, durante el método *OnGUI* de *Game* se notificará al *GUIManager* que realice su actualización y dibujado.

El orden y la creación de los manejadores puede ser expandido creando nuevas posibilidades.

Para la distribución de eventos, la clase *Game* mantendrá un acceso estático a la instancia principal de *Game*, que permitirá encolar los nuevos eventos ocurridos, para ser notificados en la próxima iteración. Por ello, todos los eventos serán almacenados en una cola dentro de *Game*, que será vaciada mediante la distribución de cada evento, a lo largo de cada iteración del juego.

Dado que una de las piezas clave de este proyecto es la comunicación por eventos, en el siguiente subcapítulo trataremos su implementación. 100

9.- Implementando eventos del juego

Un evento del juego es la mínima expresión de comunicación de *IsoUnity*. Garantiza que su distribución será conocida por todos los gestores y entidades del juego, permitiendo que cualquiera pueda generar una respuesta a dicho evento. No necesariamente todas las acciones deben ocurrir a través de eventos, pero utilizarlos garantiza el desacoplamiento de las distintas clases.

Debido a que los eventos tienes que ser guardados serán *serializables*. Como ya se comentó anteriormente, dado que no se trata de una componente la clase adecuada para crear un evento será *ScriptableObject*.

Dentro de los eventos se almacena un identificador o nombre que sirva como filtro principal para los encargados de manejarlos y un mapa de claves y valores con los distintos atributos que caracterizan el evento. Para su implementación, un mapa hash es suficiente. Sin embargo, Unity no garantiza la serialización de mapas, por lo que se mantienen dos listas con las claves y valores, para poder ser reconstruido el mapa en el momento de su deserialización. Por otro lado, se fuerza a almacenar nombres y claves en minúsculas, pese a que se reciban en cualquier formato, evitando así posibles errores en el futuro a los usuarios de la herramienta.

Uno de los problemas de serialización mencionados en el análisis reside en que Unity no provee soporte a la serialización de *System Object*. Solamente serializa *Unity Object* y los tipos básicos. Sin embargo, para almacenar en el mapa de claves y valores ambos tipos simultáneamente, es necesario utilizar *System Object*. Por ello, se ha creado un envoltorio que cataloga los tipos serializables y los almacena en las variables adecuadas. Dicho envoltorio almacena una variable para cada tipo básico. Cada vez que se escribe un valor en ella, vacía todas las variables excepto la nueva y guarda el nombre del tipo para poder acceder a él.

Pese a que, aparentemente, los eventos trabajan con *System Object*, realizan esta transformación internamente y se han dado casos aislados en los que los envoltorios no se han serializado correctamente por lo que, en revisiones futuras, será conveniente mejorar el sistema.

10.- Implementando las entidades

Se entiende por entidad todo aquel agente perteneciente a un mapa, capaz de ser reactivo a eventos, ser actualizable y desarrollar acciones a lo largo del tiempo. Para implementar su apariencia, se podrán utilizar decoraciones como base para su renderización.

La entidad entonces, es una componente añadida a un objeto perteneciente a un mapa, que permite que éste la identifique como tal, y permita el flujo de información del juego hacia ella. Una vez identificada su característica de componente, conocemos que Unity provee la clase *MonoBehaviour* para tal fin. Al ser una componente, durante la edición del mapa se encargará de mantener las entidades centradas en sus celdas y de posicionarlas en la cara superior de éstas. Fuera del modo de editor, este soporte no es forzado, por lo que se permiten los movimientos. Para identificar esto, Unity provee la clase estática *Application* que mantiene información sobre la ejecución actual y sus características.

La entidad posee unas características que la personalizan, a saber, nombre, textura para su rostro, su posición en el mapa y su capacidad para bloquear a otras entidades. Esta última se ha implementado de una manera simple, indicando si debe bloquear o no el paso a las entidades. En futuras revisiones, debería ampliarse el sistema para filtrar los tipos de entidades que permitiría que la atravesaran. Durante la especificación, se establecieron personalidades o comportamientos, en la fase de diseño, se materializaron como componentes, y en su implementación, se han desarrollado como *EntityScript*. Cada *EntityScript* es capaz de recibir eventos y actualizaciones del juego y proponer acciones de interacción con ella. Para ello, la clase *Entity* es la puerta de entrada desde el mapa, propagando todos los eventos y actualizaciones y agrupando toda la información de acciones de cada *EntityScript* que la conforme. De esta forma, con añadir nuevas componentes *EntityScript* automáticamente éstas están vinculadas al mapa y se sumarán al comportamiento de la entidad.

El *EntityScript*, dada su naturaleza, es una componente en su más pura definición y, por ello, hereda de *MonoBehaviour*. Dentro de su inicialización, buscará la *Entity* y la pondrá a disposición como variable protegida al heredero, disminuyendo así los tiempos de acceso a las características de la entidad.

Dentro de los métodos de cada *EntityScript* destacan cuatro: *EventHappened* para la recepción de eventos, *Tick* para las actualizaciones de *Game*, *Update*, para las actualizaciones temporales y *GetOptions*, para la comunicación de interacciones posibles con él. En los siguientes subcapítulos se expondrán los distintos *EntityScript* implementados destacando estos tres métodos.

11.- La componente jugador, Player

De entre todas las entidades, *player* destaca por ser aquella entidad controlable por el usuario, que le representa y responde a sus interacciones. Por ello, esta entidad cuenta con un método más, el método que recibe los eventos del controlador del juego.
Durante dicho método *player* se encarga de interpretar la información recibida por el controlador de la siguiente forma:

1. Si existe información de acciones:
    a. Si es sólo una, y ésta requiere que me mueva, se genera un evento *Move*, de la entidad que contiene a *Player*, hacia la celda donde se encuentra la entidad que generó la acción, a la distancia que él provea e indicando que se desea la respuesta de finalización del evento, para garantizar lanzar el evento contenido por la acción en el momento apropiado. De no requerir movimiento, se lanza instantáneamente.

    b. De ser múltiples acciones se crea una nueva *OptionsGUI* con la información de las distintas acciones y se da de alta en el *GUIManager*. Cuando *OptionsGUI* selecciona una correctamente, devuelve el evento de controlador con una única acción, y es procesada en el anterior caso.

2. Si existe información de celda se generará un evento *Move* hacia la celda indicada de la entidad que contiene el *player*, y se encolará.

3. Si hay datos sobre las teclas arriba, abajo, izquierda o derecha se genera un evento *Move* hacia la celda vecina que corresponda, accediendo a éstas a través de la información que el mapa provee.

En su método *EventHappened* analiza si ha ocurrido *EventFinished* del evento *Move* y durante el *Tick* se encargará de encolar el evento contenido en la acción tras haber finalizado *Move*. Antes de encolar el evento, añade el parámetro *executer* al evento,para que el receptor pueda identificar quién ejecutó la acción, permitiendo respuestas dinámicas.

12.- La componente movedora, Move

Mover es también un caso especial de entidad. Dado que aparece prácticamente en todas las entidades, no es necesario añadirla, pues *entity*, de no detectarla, la crearía y configuraría automáticamente.

Durante el método *EventHappened*, mover es susceptible a dos eventos, *Move* y *Teleport*. El segundo, al ser más prioritario, cancelará cualquier movimiento actual. Para realizarlos, dado que no podemos conocer el orden de llegada pero el segundo es más prioritario, se almacenan localmente y se procesan al llegar la actualización *Tick*.

Durante el método *Tick* de deberse teletransportar, si lo hace a una celda que esté en el mismo mapa, lo hará instantáneamente, en el caso de estar en otro mapa se pondrá visible o invisible en función del estado de mapa al que se dirija. De ser el *player*, además, marcará como activo y visible el mapa de destino. En el caso de tener que moverse, se solicitará al planificador de rutas una ruta, y de encontrarse, se comenzará a procesar dicho movimiento en *Update*.

Durante el método *Update* se solicitan las celdas al planificador de rutas y se definen los movimientos a lo largo del tiempo. Para ello existen dos tipos de movimientos: lineales y parabólicos. Los primeros, están orientados a movimientos a celdas de la misma altura o hacia rampas. Los segundos, están orientados a movimientos a celdas de distintas alturas, simbolizando forma de salto. Para implementar esto, la clase *mover* cuenta con una clase abstracta interna llamada *Movement* que almacena información del origen y destino y permite acceder a la posición actual en función de un valor entre 0 y 1, que indica el progreso del movimiento. Es clase abstracta ya que el método que proporciona la posición actual es abstracto y depende de cada implementación. En el caso lineal, se devolverá un valor proporcional a la distancia recorrida en línea recta, pero en el caso parabólico, se utilizará una función especial que dibuja dicha forma entre ambos puntos. Para generar las distintas implementaciones, *Movement* cuenta con un método estático al cual se le especifica el tipo de movimiento a generar y autoconfigura el movimiento.
Por último, de detectarse la componente decoración en la entidad y tener ésta 3 columnas y 4 filas, se identificará como un *tileset* de animación de movimiento. Durante el movimiento en este caso, se muestra la animación intercambiando el fotograma activo.

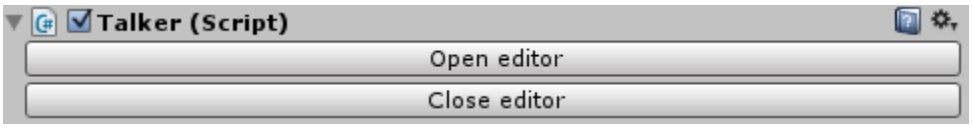
13.- La componente habladora, Talker

La capacidad de interactuar con una entidad nace de *Talker*. *Talker*, no sólo proporciona la capacidad de hablar y mostrar diálogos, sino que permite el desarrollo de secuencias completas. Para ello, sus métodos se han descrito de la siguiente forma.

Durante el método *GetOptions*, *Talker* devolverá un *array* con una acción, en cuyo interior se indica, que la entidad deberá acercarse a una distancia de 1 casilla para interactuar y que el evento es *talk* y cuenta con el parámetro *talker* que le contiene para posteriormente identificarse al recibirlo.

Durante el método *EventHappened*, se esperará a la recepción de *talk* y, al recibirlo, se guardará en una variable que en el próximo *Tick* se debe iniciar la secuencia almacenada.

Durante el método *Tick*, de haber recibido el evento, se crea un nuevo evento *start secuence* indicando como atributo *secuence* la secuencia almacenada. Dicho evento, es posteriormente capturado por el intérprete de secuencias iniciando así la interpretación.
La entidad *talker* cuenta con un Editor especial, capaz de abrir el editor de secuencias con la secuencia actual como parámetro. Dicho editor cuenta con dos botones para abrir y cerrarlo. Esto se muestra en la siguiente imagen:



14.- La componente almacén, Inventory

Para implementar el almacenamiento de objetos en una Entidad, *Inventory* cuenta con una lista de objetos y es capaz de responder a distintos eventos para añadirlos, quitarlos o utilizarlos.

Durante su *GetOptions*, se buscará una componente *player* en la misma entidad y de encontrarlo, genera una acción con el evento *open inventory*, que no requerirá moverse para ejecutarla.

Durante su *EventHappened*, interpreta los eventos *add item*, *use item* y *remove item*. Dado que el orden de llegada puede ser no homogéneo, se almacenan los eventos en listas para ser tratados a posteriori durante *Tick*. Por otro lado, de recibir el evento *open iventory* se crea una *InventoryGUI* con el inventario como argumento y se da de alta en el *GUIManager*.

Durante su *Tick*, todos los eventos recibidos conllevan operaciones sobre la lista. El orden de interpretación realiza primero las adiciones, después los usos y por último las eliminaciones. Dado que las secuencias pueden requerir del aviso síncrono para garantizar la correcta interpretación, al interpretar los eventos, se solicita a *Game* que envíe el evento *event finished* sobre cada uno de ellos.

Para implementar los objetos, se ha implementado una clase abstracta *Item* que proporciona una interfaz genérica para el tratamiento y utilización de los objetos, permitiendo a su vez su serialización automatizada. Los objetos cuentan con un nombre, una imagen que les represente y una representación en forma de *prefab* para poder instanciarse en el mapa. Cada objeto debe implementar el método *Use* que se ejecuta al ser utilizado y el método *IsEqualThan* que permite comparar los objetos entre sí, para poder agruparlos en posibles representaciones, o garantizar que no se repitan objetos catalogados como únicos.

15.- La componente objeto, ItemScript

Dado que los objetos como tal, no son entidades, es necesario que para su representación en el mapa y su interacción con el mundo, se cree un *EntityScript* que haga de envoltorio de éstos. Para ello, la clase *ItemScript* permite que los objetos puedan ser recogidos del suelo y tengan una representación.

Durante su *GetOptions*, devuelve una acción con un evento *pick*, que se ejecuta a la distancia de una casilla.

Durante su *EventHappened*, si se recibe un evento *pick*, se comprueba que el objeto es él mismo a través del parámetro *item* y, se busca un ejecutor. En unión a lo implementado en la componente jugador, al ejecutar un evento, el jugador añadirá al evento el parámetro *executer*. Una vez recibido el evento en el inventario, el parámetro *executer* realiza un enlace para la búsqueda del inventario utilizando *GetComponent*. Si se encuentra, se almacena el inventario para ser procesado durante el próximo *tick*. Se realizará un control de los eventos *event finished.* Al añadir el objeto al inventario, se verifica si dicho *event finished* ha ocurrido antes de eliminar la entidad del mapa, garantizando el traspaso del objeto del mapa a la entidad.

Durante su *Tick*, si se encuentra el inventario, se crea un evento *add item* colocando como parámetros el objeto en *item* y el inventario en *inventory*. Por otra parte, durante su *Update*, si se recibió el evento event finished correctamente, se solicita a Unity que destruya el *GameObject* que le contiene, eliminando así el envoltorio del objeto del mapa.

16.- La componente, RandomMover

Una de las formas más comunes de dar vida a una entidad es la de dar la sensación de autonomía al moverse hacia algún lugar cada cierto tiempo. Para realizar esto, si a una entidad se añade *RandomMover*, ésta puede generar aleatoriamente algún movimiento en cualquier dirección.

Para ello, solamente se hace uso del *Tick*, durante el cual, utilizando una variable de probabilidad y la variable *Time.deltaTime*, para conocer el tiempo desde la última ejecución, realizaremos un producto para calcular la probabilidad de moverse en ese instante. Dicha probabilidad se prueba generando un número aleatorio entre 0 y 1, que de ser menor que el número calculado, entra dentro de la probabilidad. Al darse como cierto, se genera un evento *move* a una de las cuatro celdas adyacentes seleccionándolas una vez más utilizando un número aleatorio.

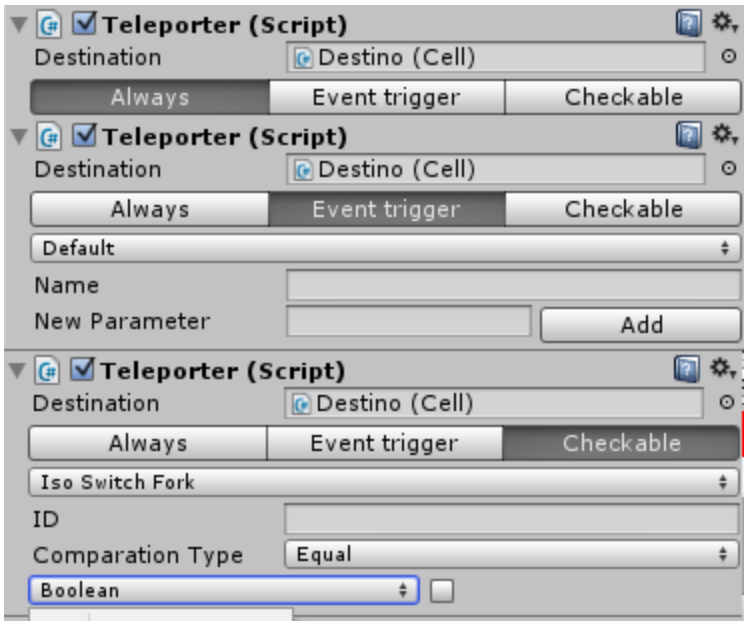
17.- La componente Teleporter

La forma de comunicar varios mapas consiste en la teletransportación de entidades entre los mapas. Para ello la componente *Teleporter* se encargará de teletransportar una entidad siempre que esté activado. Para su activación, podrá programarse siempre activo, activación al recibir un evento o utilizar las mismas bifurcaciones que las especificadas para las escenas.

Durante su *GetOptions*, creará una acción con un evento *teleport* que se ejecutará a distancia 0, para provocar que la entidad se desplace hasta él, pero sin indicar el evento. Durante su *Tick*, solicitará a su celda que le proporcione las entidades sobre ella. Iterará sobre ellas y de detectarse cualquier entidad que no sea el propio *teleporter*, se creará un evento *teleport* con la entidad y la celda de destino como argumento. Sin embargo, esto sólo se producirá si el *teleporter* está activo. Para detectarlo, si se recibiera el evento disparador o *trigger* durante *EventHappened* se marcaría como activado, o en el caso de haber seleccionado como método de activación una bifurcación, se ejecutará la bifurcación y se teletransportará a la entidad en caso de ser cierta.

Para su edición, el *teleporter* cuenta con un Editor especial, que permite seleccionar la celda de destino que estará vinculada con el *teleporter*, así como indicar el método de activación.

El modelo más simple para implementar esto fue tomar el código del editor de secuencias, y replicar la generación de editor de eventos y de editor de bifurcaciones en esta parte (se darán detalles de la implementación más adelante). Para seleccionar el tipo, se utilizará *Toolbar* de *GUILayout*. Esto se observa en la siguiente figura:



18.- Implementando los manejadores de Eventos

Dado que no sólo las entidades funcionarán con eventos, existirán sistemas a un nivel mayor capaces de manejar eventos. Durante el capítulo de Game se explicó que serían capaces de recibir eventos y actualizaciones de Game. Por ello, los manejadores de eventos implementarán la interfaz EventManager, que permitirá ambas funcionalidades. Un manejador en sí, es prácticamente un módulo para Game, un sistema que podría crear cualquier tipo de mecánica introducido en el Game a través de la comunicación por eventos.

El manejador más importante, que se implementó en primer lugar, es el manejador de mapas. Dicho manejador se encarga de realizar un puente entre Game y MapManager para realizar la distribución de eventos y actualizaciones. El segundo manejador surge de la necesidad de crear el intérprete para las secuencias. Dado que, una secuencia no es una entidad, y durante su ejecución Game se comportará de otra forma, el manejador de secuencias interpretará los eventos start secuence creando los intérpretes para las secuencias, el resto de eventos que reciba, si tuviera una secuencia iniciada, los remitiría a ella. Más adelante se explicará la implementación de la interpretación de las secuencias. Una de sus características, es que bloqueará el controlador del juego hasta la finalización de la secuencia. Para esto, tomará el delegado del controlador, dejándolo vacío provocando que todos los objetos que estuvieran registrados a las actualizaciones de controlador dejaran de recibirlas. Para realizar su interpretación, irá realizándola paso por paso durante método de actualización. Al finalizar se restablecerá el contenido del delegado.

El tercer manejador existe para gestionar la creación de emociones. Durante los capítulos anteriores apenas se mencionaron estas, pero viendo la sencillez de su implementación y la riqueza que agregaban al juego se decidió implementarlas. Para ello, el manejador interpretará el evento show emotion que contendrá, la entidad sobre la que se instanciará la emoción, y el prefab de la emoción a crear. En el momento que se detecte que la emoción se ha destruido, se enviará el event finished que avisará al manejador de secuencias que puede continuar con la ejecución.

El último manejador será el encargado de recibir los eventos de cambio de variables globales llamadas IsoSwitches y materializarlos en el archivo físico almacenado en la carpeta de recursos del proyecto. Será necesario que esté almacenado en dicha carpeta para poder ser accedido en tiempo de ejecución. Para realizar el cambio, utilizando Resources.Load lo cargará en memoria y tras esto realizará el cambio con los métodos proporcionados por la clase IsoSwitches.

19.- Implementando el controlador

Para implementar la unificación de los controles mencionada en el subcapítulo 4.4 se decidió crear una clase estática muy vinculada al ciclo de vida de la clase Game. Durante su bucle principal, Game hará llamadas explícitas para provocar eventos de controlador. Un evento de controlador no es igual que un evento de juego, pues cuenta con una estructura prefijada en la que se almacenará el estado de las entradas por parte del jugador hacia el juego. Dicha clase estática realizará el bucle principal descrito en el diseño. Para su implementación describiremos punto por punto los pasos diseñados de la siguiente forma. El primer paso consistirá en crear un objeto de la clase ControllerEventArgs en el que almacenaremos todos los valores del ratón y teclado recibidos de la clase Input de Unity. Para asegurar que recibiremos la posición del ratón pese a estarse utilizando controles de ratón, pondremos a cierta la variable Input.simulateMouseWithTouches. Tras esto, solicitaremos a GUIManager que compruebe si alguna interfaz capturara el evento de controlador. Para ello, todas las interfaces contarán con un método CaptureEvent al que se le proporcionará la variable con el estado del controlador para que la interfaz pueda identificar si debe capturarlo o no, por ejemplo, si se encuentra el ratón en su espacio.

Si alguna interfaz lo capturara, se le solicitaría que lo rellenara utilizando FillControllerEvent y pasándole como argumento la variable anterior con el estado de la entrada del usuario. Si cualquiera interfaz quisiera controlar el lanzamiento final del evento a los objetos que se encontraran escuchando al controlador, con la variable send de los ControllerEventArgs podrá modificar este comportamiento.

Por otro lado, si ninguna interfaz lo capturara, se le solicitaría a MapManager que rellenara el evento. MapManager, en este punto, si el evento de controlador indicara que se acaba de pulsar el botón izquierdo lanzaría un rayo utilizando las físicas de Unity para conocer la entidad o celda que se está tocando. Es necesario este proceso, pues el ratón se encuentra sobre la pantalla en coordenadas bidimensionales y el mapa se encuentra en coordenadas tridimensionales. De encontrarse una entidad, se le solicitarán las acciones con el método GetOptions y se rellenarán los campos entity, cell y actions del evento de controlador. De no encontrarse una entidad, pero sí una celda, se rellenará simplemente el campo cell. Si se diera cualquiera de estos casos, se marcará el evento para enviar.

Finalmente el controlador contará con un delegado público al que podrá registrarse cualquier objeto indicando el método que se llamará. Si se hubiera marcado el evento para enviar, se utilizará el delegado para avisar a todos aquellos interesados.

20.- Implementando las Interfaces

El gestor de interfaces cuenta con un mapa hash que almacena las interfaces activas y su prioridad. Basándonos en este mapa, se implementó una clase comparadora que permitía ordenar las interfaces para su tratado en función de la prioridad. Para darlas de alta, de encontrarse cualquier método iterativo se impedirá, almacenando las nuevas en una lista temporal para su posterior agregación.

Para el correcto funcionamiento con el controlador, cuenta con un método que busca por orden de prioridad la interfaz que captura el evento. Sin embargo, para su dibujado, y dado que el funcionamiento de *GUI.depth* es distinto al esperado, se reordenarán las interfaces para que las menos prioritarias sean dibujadas en último lugar, apareciendo así sobre las demás. Dado que será una clase estática, las interfaces se registrarán mediante la llamada a *addGUI* y se darán de baja mediante *removeGUI*. Para lograr el funcionamiento uniforme del bucle, todas las interfaces que se utilicen deberán de heredar de *IsoGUI* que contiene los métodos *CaptureEvent*, para conocer si capturará el evento y *Draw* para solicitar el dibujado.

21.- Interfaz de diálogos

Para implementar la interfaz de diálogos, en su constructor recibiremos el fragmento del diálogo a representar. En el fragmento van contenidos el nombre, la imagen y el texto a mostrar.

Durante el método *draw* se dibuja la interfaz haciendo uso de GUI.*Box* para delimitar el rectángulo del área del diálogo. En el interior de este rectángulo dibujaremos otra GUI.*Box* modificando la imagen de fondo del estilo para que utilice la proporcionada en el fragmento. Para dibujar el texto se utilizan dos GUI.*Label*, una para el nombre del emisor y otra para el mensaje.

Por defecto, a menos que se sobrescriba, toda *IsoGUI* capturará el evento de controlador. Durante el método *FillControllerEvent* que es llamado tras verificar que se captura el evento, se mira si el evento de controlador indica que se acaba de pulsar el botón izquierdo, y de ser así, se crea un evento *ended fragment* con el parámetro *launcher* que se obtiene en el momento de creación de la GUI se indicará quien creó la GUI. Tras encolar el evento, se solicita a *GUIManager* que destruya la GUI. Esta interfaz se observa en la siguiente imagen:

Por otro lado, la GUI de diálogos cuenta con otro constructor para opciones. Para ello, se recibirá en el constructor un *array* de *DialogOption*. Durante el método *draw*, en este caso se dividirá la altura de la pantalla entre el número de opciones y se crearán tantos GUI.*Button* como opciones existan abarcando toda la pantalla de forma vertical. De detectarse la pulsación, se generará un evento *chosen option* indicando en el parámetro

option el índice de la opción seleccionada. Además como en el caso anterior, incluiremos el parámetro *launcher*. Esta interfaz puede verse a continuación:

De esta forma, el intérprete de diálogos solamente tendrá que crear la GUI de diálogos, añadirla a *GUIManager* y esperar a recibir los eventos para continuar con la interpretación.

22.- Interfaz de inventario

La interfaz de inventario permitirá al jugador inspeccionar el inventario y realizar acciones sobre sus objetos. Para ello, en su constructor deberá recibir el inventario a representar.

Para representarlo, en primer lugar se creará una GUI.*Box* que oscurecerá la pantalla entera utilizando como referencia los valores del tamaño de la pantalla proporcionados en *Screen.width/height*. A continuación, con una GUI.*Label* se pondrá el título. En la parte inferior utilizando GUI.*Button* dibujaremos un gran botón que permita cerrar la interfaz.

Para representar los objetos, en primer lugar unificaremos los objetos iguales del inventario. Utilizaremos el método *IsEqualThan* del objeto para conocer si son iguales. Si fueran iguales, almacenaríamos en un contador cuantos objetos como él existen. Tras ello utilizando *BeginScrollView* crearemos un espacio con barra de desplazamiento vertical en el que comenzaremos a dibujar los *item* utilizando diversas GUI.*Box* para fondos y la imagen y GUI.*Label* para el nombre y la descripción. Si existiera más de un objeto de ese tipo, se dibujaría un contador con un *Label* en el interior de una *Box*.

Para lograr que se puedan realizar acciones sobre los objetos sobre cada uno y basándonos en el rectángulo que abarcan crearemos un botón invisible utilizando GUI.*Button* y la propiedad de estilo *GUIStyle.none*. De esta forma, al pulsar sobre cada objeto podremos detectarlo y en ese momento crear una interfaz de selección de acciones capaz de representar las dos acciones posibles. En su interior, las acciones almacenarán un evento *use item* para utilizarlo y otro *remove item* para tirarlo.

La representación de la interfaz y del comportamiento que se realiza al interactuar con la misma se muestra a continuación:

Gracias a su representación permite un control táctil amigable y sencillo para el usuario.

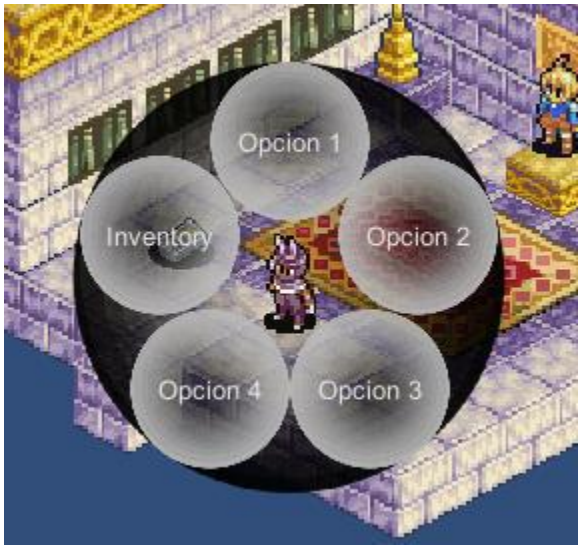
23.- Interfaz de selección de acciones

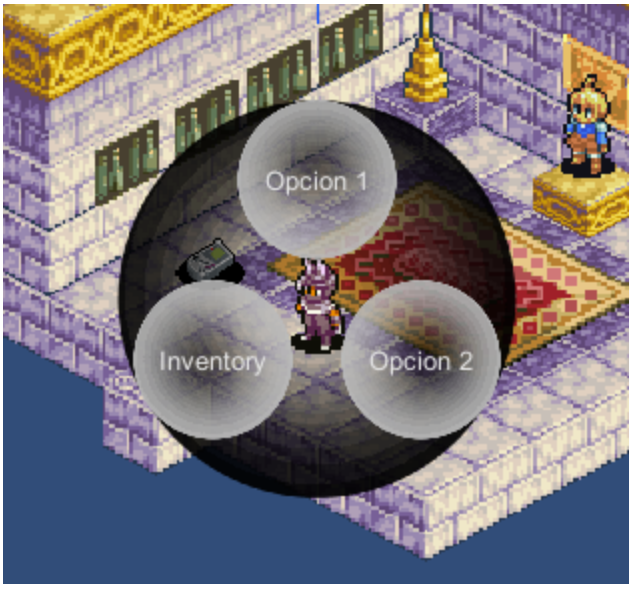
Se ha mencionado en la componente *player* que ésta, al detectar múltiples acciones del controlador deberá crear una interfaz para que el usuario decida sobre la opción que desea elegir. También, esta misma mecánica será utilizada por la interfaz del inventario para realizar acciones sobre los objetos. Dado el entorno táctil del proyecto, para facilitar la selección de la acción se realizó un diseño circular en el cual las opciones se sitúan en forma radial automáticamente distribuida en función de la cantidad de acciones.

Para implementarlo, dado que GUI no provee la capacidad para dibujar círculos, se crearon varias texturas circulares que simbolizarían el fondo y las distintas acciones a elegir. Para crear la GUI, se pasarán como parámetro la lista de acciones y el punto del mundo donde se originan.

Para dibujar el menú, se dividirán los 360º de la circunferencia entre el número de opciones. Tras esto, se realizarán los cálculos para conocer si el espacio es suficiente para dibujar las opciones, y, de no ser suficiente se expandirá el radio hasta poder contenerlas sin que se superpongan. Una vez calculado el radio, se recorrerá la circunferencia utilizando la porción de ángulo para cada opción, calculando la altura y la anchura donde colocar los botones utilizando las funciones seno y coseno. Finalmente para conocer si una acción ha sido seleccionada, si se levanta el ratón en el interior de su rectángulo se almacenará como seleccionada. Durante el próximo *FillControllerEvent*, se modificarán los atributos del *controller event* indicando la acción seleccionada como la lista de acciones, y poniendo la variable "*send"* del evento a verdadero el controlador la enviará a sus delegados.

En la siguiente imagen se muestra un menú con varias opciones:





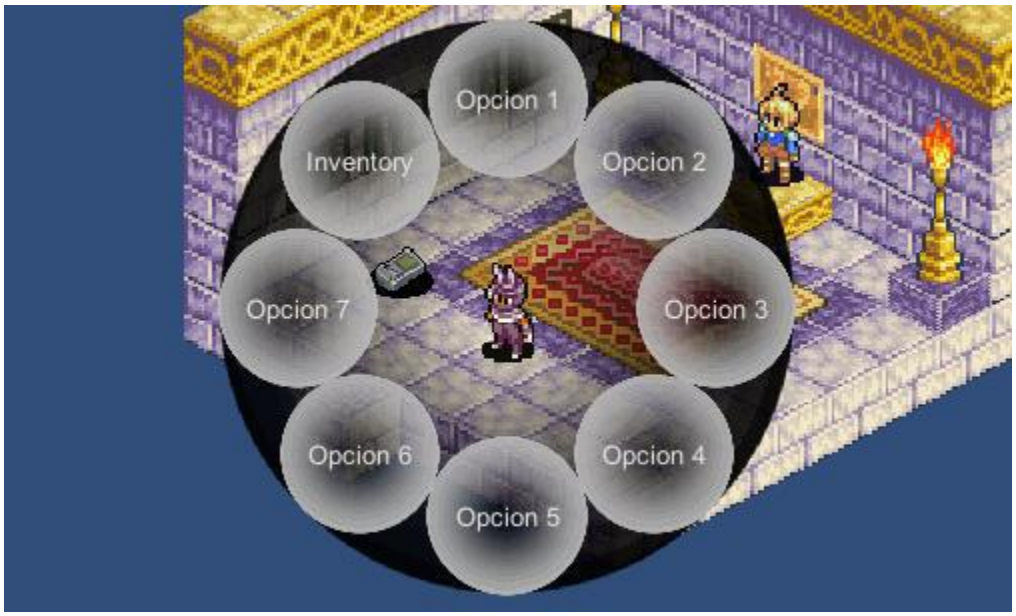


Figura 5.30 - Interfaz de selección de acciones con múltiples acciones.

En este caso, la interfaz de selección de acciones sí re implementa *CaptureEvent*, pues solamente lo capturará si el ratón se encuentra en el interior de la circunferencia. Si en ese punto se detectara el ratón fuera, se marcaría la GUI para destruir y al inicio de su próximo dibujado se solicitaría a *GUIManager* que la destruya.

24.- Interfaz de controles en pantalla

La interfaz para controles en pantalla facilitará al usuario controlar el juego simulando un teclado para entornos táctiles. Dicha interfaz, aprovechará el método *FillControllerEvent* para reconfigurarlo indicando los botones que se han pulsado. En principio, la implementación solamente abarcará los cuatro botones de movimiento para ejemplificar la creación de dicha interfaz.

Para ello, en su método *draw* se dibujarán cuatro botones utilizando para ello una técnica distinta a la utilizada por GUI.*Button*. Dicha técnica se basa en el uso del método *draw* de un estilo *GUIStyle*. Tomando como referencia un estilo personalizado se realizará la llamada a este método. Es necesaria la utilización de este método para poder marcar como activos los botones cuando se pulsen los botones de teclado correspondientes. Por ejemplo, al pulsar la tecla hacia arriba se iluminará el botón pese a no estar siendo tocado físicamente. Durante este método, además, almacenaremos en variables si se ha pulsado cualquiera de los botones. Durante el método *CaptureEvent*, si se hubiera pulsado cualquiera de los botones táctiles se retornaría que sí, para capturar el evento y recibir posteriormente la llamada de *FillControllerEvent*. Además, aprovecharemos este momento para almacenar el estado inicial de los botones del controlador y poder pintarlos en el próximo dibujado.

En el método *FillControllerEvent* se sobrescribirá el valor de los botones de teclado que no estuvieran ya pulsados con aquellos que hayan sido pulsados durante el dibujado.
El resultado final puede observarse a continuación:

25.- Implementando las secuencias

Para implementar las secuencias serán necesarias varias piezas encargadas de distribuir el almacenamiento, interpretación y edición. Para almacenarlas, se implementó una estructura en forma de árbol en la que la clase secuencia almacenaría el nodo inicial. En cada nodo tendremos una variable para almacenar el contenido del nodo y un listado de nodos hijos al nodo actual.

Dentro de los posibles contenidos que se especificaron y diseñaron, contamos con la posibilidad de almacenar eventos, diálogos y bifurcaciones. La primera es trivial, pues en el contenido se almacenará el evento en cuestión.

Para la segunda se implementó una clase diálogo que hereda de *ScriptableObject* para poder serializarla automáticamente capaz de almacenar en su interior una lista de fragmentos de diálogo y una lista de opciones. Cada fragmento podrá contener en su interior una referencia a una entidad, un nombre y un texto y de forma opcional una imagen. Las opciones por su parte contendrán un texto.

Para los últimos se implementó una clase abstracta *Checkable* que hereda de *ScriptableObject* y que contiene un método comprobar. Durante dicho método deberán realizar las comprobaciones y se retornará un valor binario. Dentro de las clases que implementan *Checkable* encontramos las mencionadas en el apartado de análisis y diseño.

Para implementar las bifurcaciones sobre variables globales se almacenará una variable para conocer el nombre de la variable global a comprobar, un tipo de comparación definido por la clase *ComparationType*, y un valor con el que comparar. Este último, utilizará la misma clase que utilizan los eventos para envolver los múltiples tipos básicos, limitando la funcionalidad para almacenar solamente tipos básicos y excluir los *UnityObject*, se utiliza la clase *IsoSwitchesManager*, que hace las funciones de singleton para el acceso al recurso donde se encuentran almacenados los *IsoSwitches*. Una vez accedido a *IsoSwitches* se accederá al valor de la variable y se comparará utilizando una distinción de casos en función del tipo de comparación seleccionado.
Para implementar las comprobaciones sobre los objetos que contiene el personaje, se almacenará una referencia al objeto a buscar y otra referencia al inventario donde buscarlo. Además se incluirá una variable binaria para indicar si se considerará cierto encontrar el objeto o no encontrarlo. Utilizando la variable de objetos del inventario y el método *IsEqualThan* de la clase *Item* se realizará la búsqueda de forma iterativa.

Dado que se anotó como requisito futuro, finalmente no se implementaron las comprobaciones sobre el tiempo. Sin embargo su especificación quedará como parte de esta memoria para poder servir de base para futuras ampliaciones del proyecto.

Una vez cerradas las implementaciones, el siguiente elemento necesario será el intérprete.

26.- Intérprete de secuencias

El intérprete de secuencias es uno de los sistemas más complejos dentro de la aplicación.

En primer lugar, la puerta de entrada para el inicio de las secuencias comenzará en el manejador de eventos llamado *SecuenceManager* o manejador de secuencias. Dicho manejador, como se ha mencionado en el apartado de manejadores, será el encargado de recibir el evento de inicio de secuencia y crear el intérprete para interpretarla.

Una vez creado un intérprete, el manejador le transmitirá todos los eventos y actualizaciones. El intérprete realizará el siguiente proceso para cada actualización. En primer lugar, comprobará que la secuencia no haya terminado. Para ello, si no tiene nodo actual, o el contenido del nodo actual es vacío se considerará por terminada. A continuación se creará un subintérprete para el contenido actual de la forma descrita más adelante. Dado que el subintérprete puede requerir varias actualizaciones o eventos para su interpretación, dicha creación sólo se producirá si no hay subintérprete asignado. Tras ello, se propagará la actualización al subintérprete para que realice su labor. Finalmente, tras su actualización se le preguntará con el método *HasFinishedInterpretation* si ha finalizado su interpretación y de ser así se solicitará al subintérprete el próximo nodo a interpretar con el método *NextNode.* Gracias a este hecho, si el subintérprete interpretara bifurcaciones u otras mecánicas podría retornar uno u otro hijo de forma transparente al intérprete. Tras adquirir el siguiente nodo se destruirá el subintérprete. Dado que algunos subintérpretes pueden ser *UnityObject*, se tratarán de destruir.

Por otro lado, el manejador de eventos transmitirá los eventos al intérprete, quien los transmitirá a su vez al subintérprete activo.

Para la creación del subintérprete adecuado para el contenido, existirá una factoría singleton con un método *createSecuenceInterpreterFor* en el cual se facilitará el nodo a interpretar. La factoría contará con un prototipo de cada subintérprete que indicará si puede interpretarlo a través del método *CanHandle*. Una vez encontrado el prototipo adecuado, se creará una copia utilizando *Clone*.

Cada subintérprete deberá implementa la interfaz *ISecuenceInterpreter* que unifica todos los métodos anteriormente mencionados para su utilización.

Dentro de los subintérpretes contaremos con tres tipos ordenados por su complejidad, el intérprete de bifurcaciones, el intérprete de eventos y el intérprete de diálogos.

El primero y más sencillo, el intérprete de bifurcaciones, simplemente ejecutará el método *Check* de la bifurcación para conocer el hijo al que acceder, y en el método *NextNode* retornará el hijo adecuado.

El segundo y ligeramente más complejo, el intérprete de eventos, encolará el evento en *Game* tan pronto reciba su primera actualización. Sin embargo, cuenta con la complejidad extra de que deberá controlar los eventos marcados como síncronos. Para ello, si se detectara el evento síncrono se esperaría a la recepción del evento *event finished* apropiado para finalizar su interpretación. Dado que los eventos no conllevan una respuesta, siempre se devolverá el primer hijo del nodo actual.

El tercero y más complejo deberá interpretar el diálogo esperando a recibir las actualizaciones de las interfaces. Para ello, comenzará creando una cola con todos los fragmentos que contenga el diálogo. Tras ello y mientras queden fragmentos en la cola extraerá un fragmento y creará una interfaz de diálogos facilitándole la información del fragmento. Además, comunicará con la clase *CameraManager* para solicitarla que enfoque a la entidad almacenada en el fragmento. La interfaz de diálogos, en el momento en el que reciba una pulsación del usuario creará un evento f*ragment ended* que indicará que dicho fragmento ya ha terminado de ser representado. Este evento será la vía para decidir cuándo extraer el próximo fragmento de la cola. Una vez se haya vaciado la cola se retornará la cámara a su objetivo original y si el diálogo tuviera al menos dos opciones, se creará una interfaz de diálogo para la selección de la opción. Ésta al detectar la pulsación de la acción enviará un evento *option selected* con el atributo *option* indicando la opción que se seleccionó. De no haber opciones, se omitirá este paso, marcando como opción seleccionada la 0. Una vez finalizada la interpretación, el nodo a retornar se elegirá en función del valor de la opción elegida.

Con esta pieza el intérprete de secuencias podrá reproducir todo tipo de secuencias y además será fácilmente expandible y reutilizable debido a su bajo acoplamiento y a la facilidad para incluir nuevos subintérpretes en la factoría.

27.- Editor de secuencias

El editor de secuencias permitirá visualizar las secuencias modificando su contenido en función de los editores disponibles. En el apartado de análisis y diseño se dieron las pautas a seguir para lograr una implementación extensible y reutilizable que se seguirán durante este subcapítulo.

En primer lugar, el editor de secuencias implementará *EditorWindow*. Esto permitirá que sea una ventana del editor, permitiendo así que pueda ser de gran tamaño y permita visualizar grandes secuencias. Para la visualización de la secuencia, basándose en el ejemplo propuesto por Linusmartensson en los foros de Unity y utilizando su librería proporcionada se creó un sistema de ventanas dentro de la ventana del editor. Para ello, utilizando el método GUI.*Window* podremos crear sub-ventanas en el interior del editor actual.

De forma recursiva se accede a todos los nodos y se asigna un id a cada nodo. Dicho id será utilizado en el momento de dibujado de la ventana para poder identificar el nodo que lo contiene. Por otro lado, se creará un rectángulo para indicar el espacio inicial que abarcará dicha ventana. Una vez dibujada la ventana, se creará una línea entre el la ventana anterior a la llamada recursiva y la ventana actual utilizando los rectángulos de las ventanas como valores para situar el origen y destino de la recta.

En el interior del dibujado de cada ventana, como se describió en el análisis y diseño, se utilizará la factoría singleton *NodeEditorFactory* para crear un editor para el nodo actual. Para ello, en primer lugar se solicitará el índice del editor adecuado para el nodo actual y se mostrará un menú desplegable utilizando EditorGUILayout.Popup para mostrar los distintos editores disponibles para el nodo. De detectarse un cambio en la selección, se eliminaría el editor actual y se crearía un nuevo editor basándose en el editor seleccionado.

Una vez creado el editor se le solicitará el dibujado y finalmente al resultado modificado del nodo. Contaremos con tres tipos de editores de nodos en función de su contenido, a saber, editor de diálogos, editor de eventos y editor de bifurcaciones. Además de éstos, existirá un editor de nodo vacío que se mostrará cuando el nodo no tenga contenido, indicando así un final de la secuencia.

En este orden se especificará la implementación de cada uno de ellos en los siguientes subcapítulos.

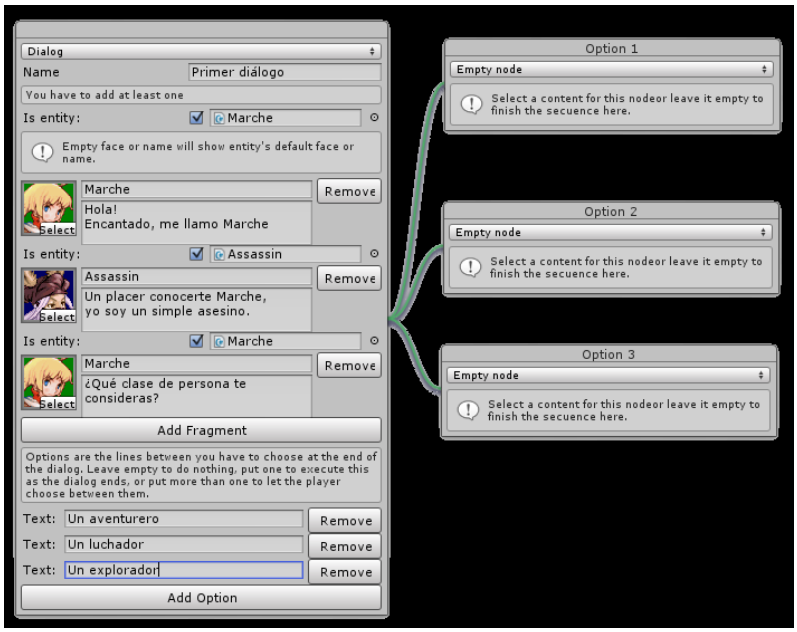
28.- Editor de nodo de diálogo

Será creado siempre que el contenido del nodo sea un diálogo. Para su representación, utilizando la clase *GUILayout* mostrará un nombre para el diálogo, el listado de fragmentos del diálogo y al final el listado de opciones.

Dentro del listado de fragmentos, se recorrerán en forma de bucle, creando para cada uno un campo para la textura de la imagen utilizando *ObjectField* indicando *Texture2D* en el tipo, un campo de texto para el nombre utilizando *TextField* y un área de texto para el contenido utilizando *TextArea.* Además se incluirá un campo ObjectField para poder seleccionar la entidad que lanzaría el fragmento. Automáticamente, de no asignarse ningún valor al nombre o a la textura, cobrarán el nombre y textura cobrarán el valor predeterminado para la entidad. Si el listado de fragmentos contara con al menos cuatro fragmentos automáticamente se convertiría en una vista con barra lateral desplazable utilizando BeginScrollView y EndScrollView. A la derecha de cada fragmento, se mostrará un botón para poder eliminarlo.

Tras los fragmentos, se mostrará el listado de opciones, indicando a su derecha un botón para eliminarlas y bajo ellas otro botón para agregarlas.
Al finalizar el dibujado, el editor comprobará el que nodo que almacena el diálogo cuenta con tantos hijos como opciones tiene el diálogo, creando o eliminando las posibles diferencias hasta igualar el número de nodos con el número de opciones, manteniendo siempre al menos un nodo hijo.

Puede verse el resultado final del editor de diálogos a continuación:



En la figura, el editor de nodo de diálogo aparece a la izquierda, y a la derecha, los nodos están editados por el editor de nodo vacío.

29.- Editor de nodo de evento

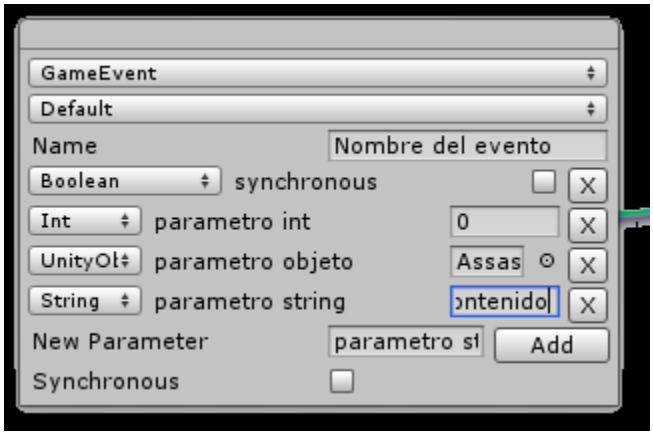
Al detectarse como contenido del nodo, un *GameEvent*, este editor será seleccionado. En su interior, como se especificó en el análisis y diseño, deberá realizar un proceso similar al del editor de secuencias para crear un editor acorde al evento que contiene.

Para ello, utilizando la factoría singleton *EventEditorFactory* solicitará los nombres de los editores actuales y buscará en ellos el nombre del evento actual. De no encontrarse, se dejará la opción 0 por defecto. A continuación se mostrarán los distintos editores utilizando un *Popup* y se creará un editor cada iteración en función del texto seleccionado. Si se detectara un cambio, se borraría el nombre del evento para evitar volver a crear el editor anterior en la siguiente iteración. Por otro lado, en el momento en el que se cree un editor, se le pasará el evento con *UseEvent* y en este punto el editor deberá cambiar el nombre del evento al que le convenga y crear en él los campos adecuados. Finalmente, se llamará al método draw para dibujar el editor y se recogerá el resultado del editor para introducirlo en el contenedor del nodo.

Dentro de los editores que la factoría de editores de evento puede crear, se buscarán todos aquellos que implementen EventEditor. Dentro de este proyecto, se han implementado los siguientes editores.

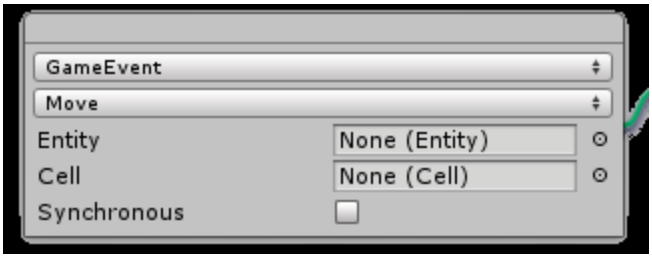
Editor por defecto:
Cuenta con una interfaz compleja para la creación de un evento, en la que se puede editar el nombre y cada parámetro así como definir los tipos de los parámetros. Es un editor bastante complejo para el usuario y su implementación requirió de la creación de un campo personalizado que permitía la selección del tipo a almacenar. Este campo, funciona en conjunto con la clase envoltorio ya mencionada para almacenar los distintos tipos básicos y crear un editor en función del tipo.

A continuación puede verse el editor por defecto:

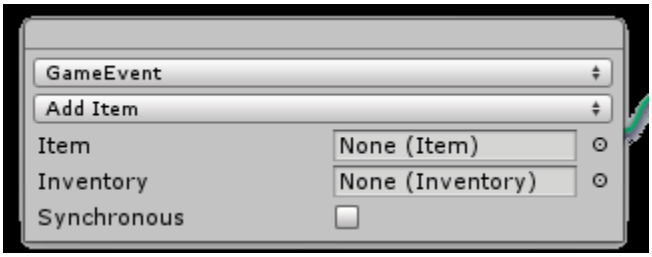


Como se puede observar es un editor bastante potente, pero que puede no ser suficiente en algunos casos, pues no permite seleccionar *assets* u objetos de un tipo específico.

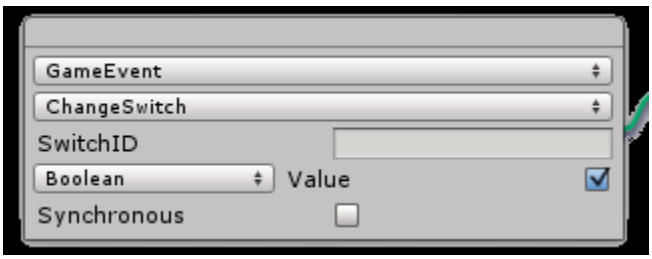
Editor de eventos *move*:
Dada la frecuencia en la que aparecerán se creó un editor para eventos *move*. Dicho editor cuenta con dos campos *ObjectField*, uno para seleccionar la celda y otro para seleccionar la entidad.
Como se puede observar, el editor personalizado es bastante más simple de manejar y el esfuerzo para su creación es mínimo, ya que consiste en rellenar una clase de apenas 20 líneas, de las cuales, sólo 2 realizan el dibujado del editor, es decir, la parte más "compleja". A continuación puede verse el resultado:



Editor de eventos *add item*:
Siguiendo el ejemplo del caso anterior y considerando la gran capacidad de ocasiones en la que necesitaremos entregar objetos, se implementó el editor para eventos *add item*. Una vez más con dos campos *ObjectField* simplificamos la creación de estos eventos.
A continuación puede verse el resultado:



Editor de eventos *change switch*:
Dado que en múltiples ocasiones los usuarios querrán cambiar los valores de sus variables globales, este editor simplificará la labor de la creación del evento apropiado.
Su resultado es aparentemente más laborioso. Sin embargo, dado que se había implementado el campo para tipos genéricos desacoplado, se pudo reutilizar con una simple llamada, haciendo este editor no mucho más complejo que los demás.
Entre sus campos cuenta con un *TextField* y el ya mencionado editor de tipos genérico *ParamEditor*. A continuación puede verse el resultado:



Con este último editor se finalizan los editores para nodos de *GameEvent*.

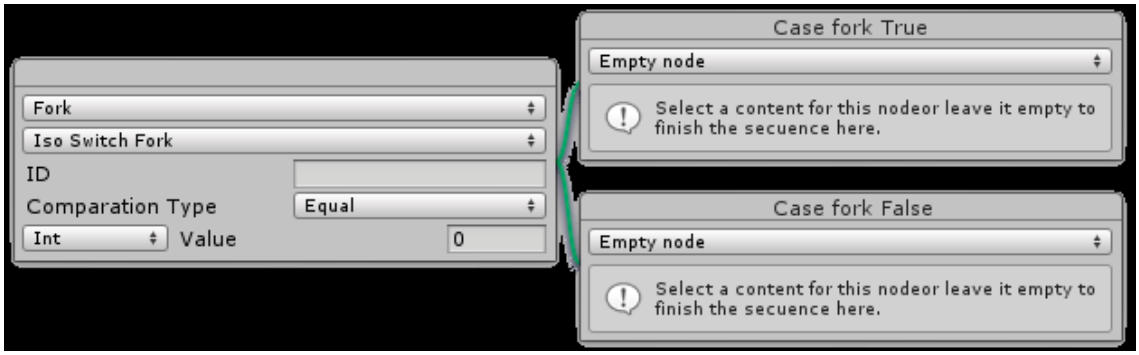
30.- Editor de nodo de bifurcación

Los nodos de bifurcaciones son aquellos que en su contenido tienen un objeto que herede de *Checkable*.

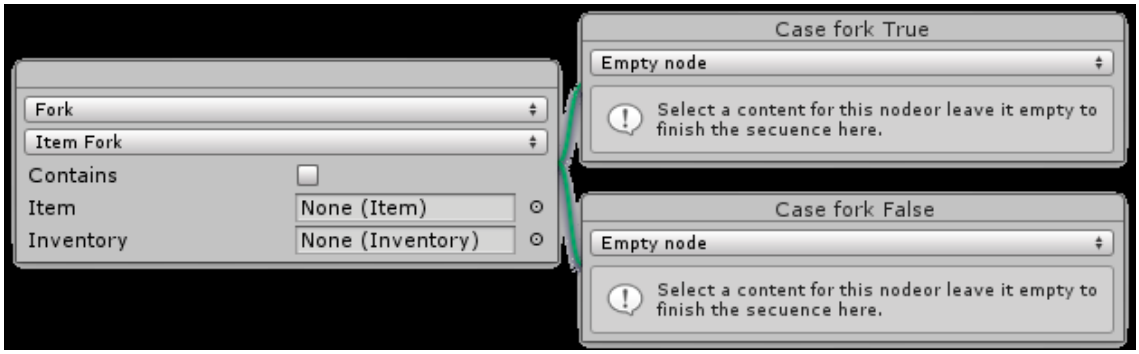
Dentro de las funciones que deberá realizar de forma genérica consistirá en asegurarse de que el nodo actual contiene dos hijos para poder elegir cada uno en función del resultado de la comprobación.

Para su creación, de manera casi idéntica a como lo realiza el editor de nodos principal, el editor de nodos de bifurcación utilizará una factoría singleton que creará un editor acorde al tipo de bifurcación a editar. Ya que esta estructura se ha explicado en apartados anteriores, pasaremos directamente a la implementación de los editores de cada tipo de bifurcación.

Editor de bifurcaciones basadas en variables globales:
Aquí se implementa el editor capaz de establecer estos tres campos utilizando para el primero *TextField*, para el segundo *EnumPopup* que permite mostrar los valores posibles del enumerado *ComparationTypes* y para el último el ya mencionado *ParamEditor* capaz de editar tipos genéricos.
El resultado de este editor puede verse a continuación:



Editor de bifurcaciones basadas en objetos de un inventario:
Para configurar el objeto y el inventario que se utilizarán para realizar la comprobación, se configurarán utilizando los campos *ObjectField* con el tipo *Item* y *ObjectField* con el tipo *Inventory*.
El resultado se observa a continuación:



Anexo 5: Análisis de los Monjes y su actividad en *La Abadía del Crimen*

A continuación se detalla el estudio realizado sobre los monjes. Para cada día y en cada franja horaria relativa a las existentes en ese día podrá verse un registro sobre la actividad. Dicho registro se ha organizado en forma de tabla para aumentar su claridad.

Día uno

Nona

Descripción

El primer día comienza con la llegada de **Guillermo** y **Adso** a las puertas de la abadía. A continuación entrarán dentro accediendo a la planta principal de ella donde les recibirá el **Abad**. Les alertara de que ha habido un asesinato y tenemos que ir corriendo a investigarlo antes de que llegue Bernardo Güi. Después nos indicará cual son nuestros aposentos.

Objetivo

LLegar a los aposentos siguiendo al abad.

Personajes

Abad

- Espera en la entrada de la iglesia para dar la bienvenida a Guillermo y a Adso, acercándose y diciendo:

"Bienvenido a esta abadía, hermano. Os ruego que me sigáis. Ha sucedido al terrible"

- De camino al pórtico nos pide que encontremos al asesino antes de que llegue Bernardo:

"Temo que uno de los monjes haya cometido un crimen. Os ruego que lo encontréis antes de que llegue Bernardo Gui, pues no deseo que se manche el nombre de esta abadía"

- De camino a las celdas les explica las normas de la abadía:

"Debéis respetar mis órdenes y las de la abadía. Asistir a los oficios y a la comida. De noche debéis estar en vuestra celda"

- Cuando llega a la celda de Guillermo:

"Esta es vuestra celda. Debo irme"

Adso

- Se encuentra junto a Guillermo en la entrada y le sigue. Podemos controlarlo.

Día uno

Vispera

Descripción

Cansados por el largo viaje, nuestros protagonistas decidieron descansar unos segundos en cuanto el abad abandonó sus aposentos. Repentinamente el repique de unas campanas les indica que ha llegado la hora de la oración. Rápidamente debemos dirigirnos a la iglesia. Guillermo se sintió algo desconcertado, debía cumplir el mandato del abad, pero desconocía donde se encontraba la iglesia. Sin embargo Adso le sacó del apuro. Conocía perfectamente la abadía y si Guillermo le seguía de cerca podría conducirle a cualquier lugar. Guillermo anotó mentalmente la habilidad de su novicio, pues era posible que en otro momento necesitara sus servicios para llevar a buen término sus investigaciones. Le siguió y en pocos segundos llegaron a la iglesia.

Allí se enfrentaron a un nuevo problema, Guillermo desconocía cuál era el lugar reservado para él. Intuyo que debía situarse dos baldosas por delante de Adso, mirando hacia el altar y en su misma línea, ya que de lo contrario el abad le sancionará.

Esperaron pacientemente que llegaran el resto de los monjes. Mientras, pudieron observaron en el particular sistema de control de la abadía, como uno de los monjes recorría zonas de la abadía desconocidas para ellos. También comprobaron asombrados como al llegar a la cocina este desaparecía y misteriosamente hacia su aparición triunfal en la iglesia por detrás del altar.
Al terminar la misa se dirigieron a su celda. Debían llegar allí ante de que lo hiciera el abad, pues de lo contrario nuevamente podría sancionarlos. Adso preguntó a su maestro si podían dormir algunas horas, Guillermo complaciente le contestó que sí. De esta forma acabó su primer día en la abadía. primer día comienza con la llegada de **Guillermo** y **Adso** a las puertas de la abadía. A continuación entrarán dentro accediendo a la planta principal de ella donde les recibirá el **Abad**. Les alertara de que ha habido un asesinato y tenemos que ir corriendo a investigarlo antes de que llegue Bernardo Güi. Después nos indicará cual son nuestros aposentos.

Personajes

Abad

- *"Oremos"*
  (Berengario, Adso, Severino y Malaquías)

Adso

- Va a misa.

Berengario

- Va a misa.

Bernardo

- Va a misa.

Malaquias

- Coge la llave del pasadizo (si está)
- Si Guillermo está en el Scriptorium:
- Le advierte que debe abandonar el edificio:
  "Debéis abandonar el edificio, hermano"
- Esperará un rato, y si Guillermo no le obedece advierte al abad:
  "Advertiré al abad"
- - Va a cerrar las puertas. Si Berengario aún no ha salido, espera a que lo haga y luego cierra.
- Si Guillermo no ha salido, cierra las puertas y va a advertir al abad.
- Si no pasa nada, va a la cocina para usar el pasadizo y llegar a misa.

Severino

- Va a misa.

Día dos

Noche

Descripción

Sigilosamente alguien entró en el aposento de Guillermo mientras este dormía para apoderarse de **las lentes**. Este solo podría recuperarlas cuando pasaran varios días.

Personajes

Abad

- Cuando consigue dejar a Guillermo en su celda, cambia la hora a NOCHE.
- Cierra la puerta que une las celdas con la iglesia.
- Se va a su celda a dormir un rato.

Adso

- Pregunta si dormimos. Si contestamos que sí, salimos de la celda o tardamos en contestar, pasamos al siguiente día. Si contestamos que no, no volverá a hacer la pregunta hasta que salgamos de la celda y entremos de nuevo.

Berengario

- Va a su celda.

Bernardo

- Va a su celda.

Malaquias

- Va a su celda.

Severino

- Se va a su celda a dormir.

Día dos

Prima

Descripción

Nuevamente el repique de las campanas despertó a Guillermo. Como el día anterior, ambos se dirigieron a la iglesia para la oración, situándose en el lugar reservado para ellos.(foto en la iglesia). Durante su estancia en la abadía esta acción se repetirá cada mañana, situándose siempre en la posición exacta, para no ser sancionados.

Todo estaba preparado para el sermón, cuando el abad anunció el descubrimiento del cadáver de Berengario, uno de los monjes traductores de la abadía. Pocos segundos después el abad llamó a Guillermo, quien tras encontrarle escuchó atentamente sus palabras.

Personajes

Abad

- *"Hermanos, Venancio ha sido asesinado"*
  (Berengario, Adso, Severino y Malaquías)

Adso

- Va a misa.
  "Debemos ir a la iglesia, maestro"

Berengario

- Va a misa.

Bernardo

- Va a misa.

Malaquias

- Va a misa.

Severino

- Va a misa.

Día dos

Tercia

Descripción

Guillermo y Adso aprovecharon el breve descanso de la hora tercia para recorrer la abadía y memorizar la localización exacta de las estancias.

La excursión les condujo a la biblioteca donde encontraron y recogieron una llave que posteriormente le servirá para abrir el pasadizo secreto que recorrerían por la noche, pues estaba rigurosamente prohibido acceder a él. La llave estaba custodiada por el bibliotecario, pero Guillermo haciendo uso de la astucia que le caracterizaba le distrajo colocándose cerca de la barandilla del patio, mirando hacia otro lado y guiando a Adso para que este se dirigiera por detrás de la mesa hasta el lugar donde se encontraba la llave.

Con la llave en su poder se dirigieron al comedor. Por el camino observaron un curioso pergamino y un libro encima de un escritorio, pero no pudieron recogerlos ta que estaban vigilados.

Personajes

Abad

- Llama a Guillermo y le explica que la biblioteca es un lugar secreto:
  "Venid aquí, Fray Guillermo"
  "Debéis saber que la biblioteca es un lugar secreto. Sólo Malaquías puede entrar. Podéis iros"

Berengario

- Va a vigilar la mesa de Venancio.

Malaquias

- Va a su mesa del scriptorium, y deja la llave del pasadizo (si la tiene).

Día dos

Sexta

Descripción

Llegaron al comedor. Allí Guillermo debía situarse junto a la segunda columna por la izquierda. Este punto le supuso a Guillermo un fuerte desgaste de su contador, ya que mientras no consiguiera encontrar el lugar exacto, este descendería vertiginosamente.

Después de comer se dirigieron hacia la cocina donde encontraron una de las dos entradas del pasadizo secreto. Pero rápidamente las campanas les indicaron que deberían dirigirse a la iglesia.

Después regresaron a la celda. Adso quería dormir, pero Guillermo con un rotundo no le indico que debían proseguir las investigaciones. Corrían el riesgo de que el abad les descubriera fuera de su aposento, pero merecía la pena arriesgarse. Guillermo había elaborado una estrategia.

Personajes

Abad

- Espera a que lleguen al comedor Berengario, Adso y Severino

Adso

- Va al refectorio.
  "Debemos ir al refectorio, maestro"

Berengario

- Va al comedor.

Bernardo

- Va al refectorio.

Malaquias

- No va al refectorio. Se queda en el scriptorium vigilando.

Severino

- Va al refectorio.

Día dos

Nona

Personajes

Berengario

- Va a vigilar la mesa de Venancio.

Vispera

Personajes

Malaquias

- Coge la llave del pasadizo (si está)
- Si Guillermo está en el Scriptorium:
- Le advierte que debe abandonar el edificio:

"Debéis abandonar el edificio, hermano"

- Esperará un rato, y si Guillermo no le obedece advierte al abad:

"Advertiré al abad"

- Va a cerrar las puertas. Si Berengario aún no ha salido, espera a que lo haga y luego cierra.
- Si Guillermo no ha salido, cierra las puertas y va a advertir al abad.
- Si no pasa nada, va a la cocina para usar el pasadizo y llegar a misa.

Severino

- Va a misa.

Abad

- *"Oremos"*
  (Berengario, Adso, Severino y Malaquías)

Adso

- Va a misa.
  "Debemos ir a la iglesia, maestro"

Berengario

- Va a misa.

Bernardo

- Va a misa.

Completas

Personajes

Abad

- Va a la puerta de la celda de Guillermo y espera a que entren.
- Tras esperar un rato, si Guillermo no entra, le ordena que lo haga (va a buscarle si es necesario)
- Cuando consigue dejar a Guillermo en su celda, cambia la hora a NOCHE

Adso

- Va a su celda.

Berengario

- Va a su celda.

Bernardo

- Va a su celda.

Malaquias

- Va a su celda.

Severino

- Va a su celda.

Día tres

Noche

Descripción

Rápidamente se encaminaron hacia una puerta secreta que se encontraba en la habitación posterior al altar, por la que unos días antes había aparecido el monje mientras esperaban la llegada del resto de los monjes para la oración, Entraron por ella y misteriosamente aparecieron detrás de la chimenea de la cocina. Esta es la única forma de llegar a la biblioteca, ya que por las noches todas las puertas permanecía cerrada.

Subieron a la biblioteca y buscaron el pergamino y el libro que habían encontrado el día anterior. Sobre el escritorio hallaron el pergamino, se apoderaron de él y comprobaron que el libro había desaparecido.

Regresaron a la celda rápidamente. Para que el abad no les sorprendiera decidieron aguardar hasta que amaneciera en la puerta del pasadizo, esta debía quedarse abierta para que pudieran ocultarse.

Personajes

Abad

- Cuando consigue dejar a Guillermo en su celda, cambia la hora a NOCHE.
- Cierra la puerta que une las celdas con la iglesia.
- Se va a su celda a dormir un rato.

Adso

- Pregunta si dormimos. Si contestamos que sí, salimos de la celda o tardamos en contestar, pasamos al siguiente día. Si contestamos que no, no volverá a hacer la pregunta hasta que salgamos de la celda y entremos de nuevo.

Berengario

- Se pone la capucha y sale de su celda destino a las escaleras sureste que dan al scriptorium. Una vez allí, sube al scriptorium y coge el libro. Luego se dirige a la celda del herbolario (está envenenado y va a buscar algo que calme sus dolores). Cuando llega, amanece.

Bernardo

- Va a su celda.

Malaquias

- Va a su celda.

Severino

- Se va a su celda a dormir.

Día tres

Prima

Descripción

Se dirigieron hacia la iglesia. El abad sin ocultar ya su preocupación les comunicó la desaparición del ayudante del bibliotecario.

Personajes

Abad

- *"Hermanos, Berengario ha desaparecido. Temo que se haya cometido otro crimen"* (Adso, Severino y Malaquías)

Adso

- Va a misa.
  "Debemos ir a la iglesia, maestro"

Jorge

- Aparece en el lugar donde será presentado a Guillermo.

Bernardo

- Va a misa.

Malaquias

- Va a misa.

Severino

- Va a misa.

Día tres

Tercia

Descripción

Al terminar la oración, el abad decidió presentar a Guillermo al más anciano de los monjes de la abadía, Jorge. Este airadamente les hablo de la presencia del anticristo en la abadía.

Personajes

Abad

- Llama a Guillermo para presentarle a Jorge:
  "Venid aquí, Fray Guillermo"
  "Quiero que conozcáis al hombre más viejo y sabio de la abadía"
- Tras seguirle y llegar hasta donde está Jorge:
  "Venerable Jorge, el que está ante vos es Fray Guillermo, nuestro huésped"

Jorge

- Al ser presentado a Guillermo por el abad:
  "Sed bienvenido, venerable hermano; y escuchad lo que os digo. Las vías del anticristo son lentas y tortuosas. Llega cuando menos lo esperas. No desperdicieis los últimos días"

Malaquias

- Va a su mesa del scriptorium, y deja la llave del pasadizo (si la tiene).

Sexta

Personajes

Abad

- Espera a que lleguen al comedor Berengario, Adso y Severino

Adso

- Va al refectorio
  "Debemos ir al refectorio, maestro"

Jorge

- Se va a su celda.

Bernardo

- Va al refectorio.

Malaquias

- No va al refectorio. Se queda en el scriptorium vigilando.

Severino

- Va al refectorio.

Día tres

Nona

Descripción

Guillermo y su inseparable Adso después de ir al comedor y obedecer así las órdenes del abad, se dirigieron a la cocina. Allí recogieron la lámpara de aceite imprescindible para la excursión nocturna que Guillermo había planeado.

Personajes

Malaquias

- Continúa vigilando la entrada de la biblioteca.

Severino

- Se da paseos.

Completas

Personajes

Abad

- Va a la puerta de la celda de Guillermo y espera a que entren.
- Tras esperar un rato, si Guillermo no entra, le ordena que lo haga (va a buscarle si es necesario)
- Cuando consigue dejar a Guillermo en su celda, cambia la hora a NOCHE

Adso

- Va a su celda.

Bernardo

- Va a su celda.

Malaquias

- Va a su celda.

Severino

- Va a su celda.

Día cuatro

Noche

Descripción

Dispuestos a investigar la localización exacta del laberinto, decidieron no dormir. Esta primera visita les permitió aprender a guiarse por él.

Personajes

Abad

- Cuando consigue dejar a Guillermo en su celda, cambia la hora a NOCHE.
- Cierra la puerta que une las celdas con la iglesia.
- Se va a su celda a dormir un rato.

Adso

- Pregunta si dormimos. Si contestamos que sí, salimos de la celda o tardamos en contestar, pasamos al siguiente día. Si contestamos que no, no volverá a hacer la pregunta hasta que salgamos de la celda y entremos de nuevo.

Bernardo

- Va a su celda.

Malaquias

- Va a su celda.

Severino

- Se va a su celda a dormir.

Día cuatro

Tercia

Descripción

Guillermo y Adso se dirigieron a la biblioteca, por el camino encontraron al monje encargado del herbolario, quien amablemente le informo del resultado de la autopsia practicada en el cadáver encontrado. Lo más destacado de su informe era la aparición de unas misteriosas manchas en la lengua y en los dedos.

Personajes

Abad

- Llama a Guillermo para decirle que ha llegado Bernardo Gui
  "Venid aquí, Fray Guillermo"
  "Ha llegado Bernardo, debéis abandonar la investigación"

Malaquias

- Va a su mesa del scriptorium, y deja la llave del pasadizo (si la tiene)..

Severino

- Busca a Guillermo para hablarle sobre la autopsia de Berengario:
  "Es muy extraño, hermano Guillermo. Berengario tenía manchas negras en la lengua y en los dedos"

Día cuatro

Sexta

Personajes

Abad

- Espera a que lleguen al comedor Adso y Severino.

Adso

- Va al refectorio
  "Debemos ir al refectorio, maestro"

Bernardo

- Aparece en la entrada de la iglesia.

Malaquias

- No va al refectorio. Se queda en el scriptorium vigilando.

Severino

- Va al refectorio.

Nona

Descripción

Bernardo Güi llego a la abadía. Tras reponer energía en el comedor, Bernardo haciendo buen uso de los poderes que el abad le había otorgado, exigió a Guillermo el pergamino con la intención de examinarlo. Inmediatamente, aunque contrariado, Guillermo le entrego el manuscrito.

Personajes

Bernardo

- Va a su celda.

Malaquias

- Continúa vigilando la entrada de la biblioteca.

Severino

- Se da paseos.

Día cuatro

Visperas

Descripción

Guillermo y Adso se dirigieron a la biblioteca, por el camino encontraron al monje encargado del herbolario, quien amablemente le informo del resultado de la autopsia practicada en el cadáver encontrado. Lo más destacado de su informe era la aparición de unas misteriosas manchas en la lengua y en los dedos.

Personajes

Abad

- *"Oremos"*
  (Bernardo, Adso, Severino y Malaquías)

Adso

- Va a misa.
  "Debemos ir a la iglesia, maestro"

Bernardo

- Va a misa.

Malaquias

- Coge la llave del pasadizo (si está)
- Si Guillermo está en el Scriptorium:
- Le advierte que debe abandonar el edificio:
  "Debéis abandonar el edificio, hermano"
- Esperará un rato, y si Guillermo no le obedece advierte al abad:
  "Advertiré al abad"
- Va a cerrar las puertas. Si Berengario aún no ha salido, espera a que lo haga y luego cierra.
- Si Guillermo no ha salido, cierra las puertas y va a advertir al abad.
- Si no pasa nada, va a la cocina para usar el pasadizo y llegar a misa.

Severino

- Va a misa.

Completas

Personajes

Abad

- Va a la puerta de la celda de Guillermo y espera a que entren.
- Tras esperar un rato, si Guillermo no entra le ordena que lo haga (va a buscarle si es necesario)
- Cuando consigue dejar a Guillermo en su celda, cambia la hora a NOCHE

Adso

- Va a su celda.

Bernardo

- Va a su celda.

Malaquias

- Va a su mesa del scriptorium, y deja la llave del pasadizo (si la tiene)..

Severino

- Va a su celda.

Día cinco

Noche

Descripción

Prosiguiendo sus investigaciones y encontraron una llave olvidada por el abad junto al altar. La recogieron y regresaron a gran velocidad a su celda para no ser sorprendidos por este.

Personajes

Abad

- ″Si Guillermo coge la llave del altar, el abad se despierta
  - Si está despierto:
  - Si Guillermo está en su celda, o en el ala izquierda de la abadía, se vuelve a dormir (menos tiempo)
  - De lo contrario va a buscarle. Si Guillermo llega a la cocina a través del pasadizo mientras le buscan, vuelve a su celda a dormir.

Adso

- Pregunta si dormimos. Si contestamos que sí, salimos de la celda o tardamos en contestar, pasamos al siguiente día. Si contestamos que no, no volverá a hacer la pregunta hasta que salgamos de la celda y entremos de nuevo.

Bernardo

- Va a su celda.

Malaquias

- Va a su celda.

Severino

- Se va a su celda a dormir.

Día cinco

Prima

Descripción

El padre herbolario dispuesto a colaborar con Guillermo momentos antes de cumplir con la oración le comunicó la aparición de un extraño libro en su escritorio. Por fin habían encontrado el libro.

Personajes

Abad

- *"Oremos"*
  (Bernardo, Adso, Severino y Malaquías)

Adso

- Va a misa.
  "Debemos ir a la iglesia, maestro"

Bernardo

- Va a misa.

Malaquias

- Va a misa

Severino

- Si Guillermo no está en el ala izquierda de la abadía, va a buscarle y le cuenta que ha encontrado el libro en su celda.
  "Escuchad hermano, he encontrado un extraño libro en mi celda"

Día cinco

Tercia

Descripción

Mientras el abad entretenía a Guillermo, el bibliotecario mató al herbolario y lo encerró en su habitación son su propia llave. El libro desapareció nuevamente.

Personajes

Abad

- Llama a Guillermo para decirle que Bernardo se marcha:
  "Venid aquí, Fray Guillermo"
  "Bernardo abandonará hoy la abadía"

Bernardo

- Se va de la abadía. Va hasta las escaleras de la entrada (donde comienza el juego) y desaparece.

Malaquias

- Va a la celda del herbolario y mata a Severino. Luego va a su mesa.

Severino

- Se queda en su celda esperando a que le maten.

Día cinco

Sexta

Personajes

Abad

- Espera a que lleguen al comedor Adso.

Adso

- Va al refectorio
  "Debemos ir al refectorio, maestro"

Nona

Descripción

Todo estaba dispuesto para comer, cuando el abada y Guillermo se dieron cuenta que el hermano herbolario no había acudido a la cita diaria. Se dirigieron a su celda y encontraron allí el cadáver. Mientras tanto, el bibliotecario aprovechando la conmoción general encerró el libro en la habitación secreta del laberinto.

Personajes

Abad

- Al terminar la comida nos pide que busquemos a Severino:
  "Venid, Fray Guillermo, debemos encontrar a Severino"
- Tras llamar a la puerta de la celda de Severino:
  "*Dios santo... han asesinado a Severino y le han encerrado"*
  (Al terminar de decirlo se pasa a Vísperas)

Malaquias

- Continúa vigilando la entrada de la biblioteca.

Día cinco

Visperas

Descripción

El bibliotecario decidió ojear el libro. Moribundo consiguió llegar a la iglesia, pero por el camino perdió las lentes de Guillermo y las llaves robadas. Una vez en la capilla pronunció sus últimas palabras, muriendo a los pocos segundos.

Personajes

Abad

- *"Malaquías ha muerto"*
  (Adso y Malaquias)

Adso

- Va a misa.
  "Debemos ir a la iglesia, maestro"

Malaquias

- Nada más llegar a su sitio en la iglesia pronuncia estas palabras y muere:
  "Era verdad, tenía el poder de mil escorpiones"

Completas

Personajes

Abad

- Va a la puerta de la celda de Guillermo y espera a que entren.
- Tras esperar un rato, si Guillermo no entra, le ordena que lo haga (va a buscarle si es necesario)
- Cuando consigue dejar a Guillermo en su celda, cambia la hora a NOCHE

Adso

- Va a su celda.

Día seis

Noche

Descripción

Guillermo y Adso decidieron continuar sus pesquisas. Al llegar a la biblioteca sobre un escritorio encontraron la llave perdida y en el torreón noroeste del laberinto las lentes. Con estos objetos en su poder regresaron la llave perdida y en el torreón noroeste del laberinto las lentes. Con estos objetos en su poder regresaron a la celda, sin olvidar lámpara de aceite, como cada día.

Prima

Personajes

Abad

- *"Oremos"*
  (Adso)

Adso

- Va a misa.
  "Debemos ir a la iglesia, maestro"

Tercia

Descripción

Guillermo continúo investigando y sus pesquisas le condujeron a la habitación del padre herbolario, donde encontró unos guantes que recogería para utilizarlos más tarde.

Personajes

Abad

- Llama a Guillermo para decirle que mañana debe abandonar la abadía:
  "Venid aquí, Fray Guillermo"
  "Mañana, abandonaréis la abadía"

Día seis

Sexta

Personajes

Abad

- *"Oremos"*
  (Adso)

Adso

- Va a misa.
  "Debemos ir a la iglesia, maestro"

Nona

Descripción

Utilizando la llave que el abad dejo olvidada en el altar, Guillermo y Adso llegaron a la celda de este. Allí recuperaron el manuscrito que contenía la clave para atravesar el espejo del laberinto que encerraba la habitación secreta.

Visperas

Personajes

Abad

- *"Oremos"*
  (Adso)

Adso

- Va a misa.
  "Debemos ir a la iglesia, maestro"

Completas

Personajes

Abad

- Va a la puerta de la celda de Guillermo y espera a que entren.
- Tras esperar un rato, si Guillermo no entra le ordena que lo haga (va a buscarle si es necesario)
- Cuando consigue dejar a Guillermo en su celda, cambia la hora a NOCHE

Adso

- Va a su celda.

Día siete

Prima

Personajes

Abad

- *"Oremos*
  (Adso)

Adso

- Va a misa.
  "Debemos ir a la iglesia, maestro"

Tercia

Personajes

Abad

- Llama a Guillermo para decirle que debe abandonar la abadía:
- *"Venid aquí, Fray Guillermo"*
- *"Debéis abandonar ya la abadía"*

Día cinco

Visperas

Descripción

Con la lámpara de aceite recargada, se encaminaron hacia la misteriosa y escondida habitación secreta, Al llevar a ella encontraron el espejo. Se situaron lo más cerca posible de el en las escaleras del centro y recordaron la leyenda del manuscrito. La clave en la primera y última letra de la palabra *quatuor*. Pulsando la Q y la R el espejo misteriosamente desapareció.

En la estancia encontraron a Jorge, el anciano monje ciego. Este se dirigió a Guillermo cogió el libro prohibido y escuchó atentamente la historia que sobre él le contó el anciano invitándole a leerlo. Era un libro de Aristóteles prohibido durante años. Guillermo quien previamente se había colocado los guantes le ojeo y lo comprendió todo. Sus páginas estaban envenenadas y cuando alguien utilizaba el pulgar humedecido el veneno acababa con la curiosidad del ávido lector.

El anciano ciego desapareció por la puerta, raudos Guillermo y Adso le siguieron para no perderle de vista y entonces… Sucedió lo inevitable. Jorge se empieza a comer el libro y como consecuencia de ello muere envenenado al igual que los monjes que leyeron el libro.

Personajes

Adso

- Explica a Jorge que Guillermo lleva guantes:
  "Venerable Jorge, vos no podéis verlo, pero mi maestro lleva guantes. Para separar los folios tendría que humedecer los dedos en la lengua hasta que hubiera recibido suficiente veneno"
- Al descubrir a Jorge comiéndose el libro:
  "Se está comiendo el libro, maestro"

Jorge

- Al entrar Guillermo en la habitación que hay tras el espejo, dice:
  "Sois vos, Guillermo.... Pasad, os estaba esperando"
- Al terminar la frase deja el libro sobre la mesa y dice:
  "Es el Coena Cipriani de Aristóteles. Ahora comprenderéis porque tenía que protegerlo. Cada palabra escrita por el filósofo ha destruido una parte del saber de la cristiandad. Sé que he actuado siguiendo la voluntad del Señor... Leedlo, pues, Fray Guillermo. Después te lo mostraré a ti muchacho"
- Tras explicarle Adso que Guillermo lleva guantes, dice:
  "Fue una buena idea, ¿Verdad? Pero ya es tarde"
- Mientras Jorge habla, le quita el libro a Guillermo, apaga la luz y huye de la habitación.
- Espera en la habitación iluminada del noroeste de la biblioteca, comiéndose el libro.